Graphical Display of a Missile Endgame Scenario

THESIS

Joseph E. Moritz, Lieutenant, USAF

AFIT/GCS/ENG/96D-20

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY
AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT/GCS/ENG/96D-20

Graphical Display of
a Missile Endgame Scenario

THESIS

Joseph E. Moritz, Lieutenant, USAF

AFIT/GCS/ENG/96D-20

AFIT/GCS/ENG/96D-20

Graphical Display of a Missile Endgame Scenario

THESIS

Presented to the Faculaty of the Graduate School of Engineering

of the Air Force Institue of Technology

Air University

In Partial Fulfillment of the

Requirements for the Degree of

Master of Science (Computer Systems)

Joseph Edward Moritz, B.A.

Lieutenant, USAF

December, 1996

Acknowledgments

Don't Panic!

- Douglas Adams

This quote has rung in my ears throughout my time here at AFIT. Furthermore, it sums up the AFIT experience. To those of you reading this thesis in future years, remember to keep this quote in mind while you toil in your mountains of seemingly endless work.

With this in mind, I would like to express my appreciation to Major Keith Shomper for his unquenchable thirst for knowledge. His thirst not only found this thesis topic but also rescued my linguistic rules from the clutches of the Microsoft Grammar Checker. Next on the list of accolades is Kevin McCardle and Doug McCown without whom I would have been drowning in a sea of trajectories and cosines, translations and polar angles. Their experience is a river of knowledge that they let me drink from often. Finally, special recognition goes to Captain Jeff Bush for his work and enthusiasm brought to this effort. Until, of course, he realized that he couldn't do my thesis for me because he had his own work to do.

I would especially like to thank my best friend Anna for helping me through the late nights and difficult times here at AFIT. I will never be able to repay my debt to her.

Joseph E. Moritz

Table of Contents

Page

Page

Page

List of Figures

Figure Page

Figure Page

List of Tables

AFIT/GCS/ENG/96D-20

Abstract

Traditionally, computer programs to model and simulate air-to-air missile endgame engagements have focused on improving the underlying probability of kill models. This focus, however, understates the benefits of current computer graphics technology for visualization and direct manipulation of missile endgame parameters and building engagement understanding through real-time, three dimensional simulation of the endgame. Our research has been to develop a versatile simulation system to display the missile endgame parameters and allow graphical interaction with these parameters. This program also allows the user to view animated engagements of previously designed endgames. The current project scope is to provide user feedback of multiple fixed cone fuzes, warhead fragment trajectories, and target damage incurred by warhead detonation given a particular set of missile and target endgame encounter parameters. Our implementation, the AFIT Missile Endgame Simulator (AMES), achieves its goals using a Motif/Inventor graphical user interface to change engagement parameters through both text input and direct object manipulation. It also allows these engagements to be animated over time to help develop intuition about the objects' interactions.

Graphical Display

of

a Missile Endgame Scenario

1. *Introduction*

1.1. Background

To implement its operational goals to fly, fight, and win, the Air Force must continually improve and advance its various weapon system technologies. However, keeping Air Force operations on the forefront, in a diversified and continuously changing technological environment, has become a greater challenge. To face this task, the Air Force has supported efforts to use computer based modeling and simulation to produce next generation weapon systems.

Modeling and simulation are powerful tools for predicting accurate results of real world events. The Air Force currently employs computer based modeling and simulation to analyze large-scale, computer generated land battles, air-to-air and air-to-ground combat using flight simulators, and probability of kill statistics for missile endgame scenarios among many other activities. By producing these simulation systems, the Air Force takes advantage of a new and continually improving technology to produce more accurate and reliable weapons and to better train its personnel.

During the 1990's, the Air Force funneled most of the computer modeling and simulation budget to flight simulators, while the other areas of computer modeling and simulation were starved. For example, the Air Force developed most missile endgame simulation programs (e.g., FASTGEN, COVART and SHAZAM), during the 1970's and mid-1980's. These programs are still the basis for most military, graphical endgame simulations. Despite a funding drought during recent years, the Air Force has supported development of new programs to improve on these software packages by modeling the endgame more realistically. One such recent development is the Ordnance Package Evaluation Code program (OPEC).

OPEC simulates an endgame scenario and calculates a probability of kill of a missile hitting near or on a target for a set of input parameters. To calculate a probability of kill that is within a specified confidence interval, OPEC runs the simulation multiple times and employs a Monte Carlo method to generate a probabilistic result (4:2-8). Compared to previous probability of kill calculation programs, such as SHAZAM, OPEC improves on the missile endgame's computational model. However, a drawback of OPEC is it does not provide scaled time, graphical simulation of the endgame. This lack of graphical feedback hinders the ability of missile systems engineers to analyze the effects of new engineering requirements for missile systems. Since they are unable to see the simulation events as they unfold, they must make their decisions based on the non-intuitive tables of data output by OPEC.

A graphical output of the missile simulation would give engineers a tremendous advantage in developing better weapon systems. Given a graphical simulation, engineers can make better predictions for improvements of both offensive and defensive methods. For example, an improvement in warhead yield may be discovered when given a graphical simulation of a newly devised fragmentation pattern (2:1). Graphical simulation will also save time and money by reducing the number of live fire or hardware-in-the-loop (HIL) simulations the engineers need to run to discover and test these new concepts (3).

1.2. Program requirements.

A graphical software package to fit the need to improve graphical missile endgame feedback should allow an engineer to create configurable simulations. The engineer should have a choice of various operational missiles to add to the simulation including the AMRAAM and the Aim-9 and a choice of targets, including the Tu-22M Backfire, the Tu-160 Blackjack, and the Tu-95 Bear. The system should also give the engineer the flexibility to position the missile and target in any orientation within a three dimensional environment. Given these initial setup parameters, the program should simulate the interaction between the target and missile models according to the given velocity vector applied to each. With these capabilities, an engineer can then create different simulations depending on the desired needs.

To allow for backward compatibility with current software in use, the system should also use the output files created by OPEC as input data for a simulation scenario as well as data formatted in WL/MNMF's ENCOUNT format (12). Depending on these input variables, the program should show how the missile's probability of kill determinants interact with the target. These determinants include configuration of the missile's fuze sensor, the missile and target state configurations, and warhead blast fragmentation. These additional capabilities should be met to ensure that missile system engineers can use current simulation system data to predict and test future ideas and requirements.

1.3. Problem Definition

I propose to create a state-of-the-art software program that will allow missile engineers to either create an endgame scenario using a mouse interface or view OPEC or ENCOUNT data in a scaleable time graphical form. The creation of this program involves the design and implementation of multiple components required by the sponsor, WL/MNMF, Eglin Air Force Base, Florida. These components include

1. the design and creation of a graphic interface to display the missile and target models,
2. a menu interface that allows the user to control the simulation and change its attributes,
3. the inclusion of velocity vectors that allows the user to change the missile's and target's direction and speed,
4. a control pad that allows the user to play, fast forward, rewind or pause the simulation as desired,
5. graphical simulation of the blast fragmentation and dispersion around a predetermined burst point, and
6. simulation of multiple fixed cone fuze sensor coverage patterns.

1.4. Summary of Current Knowledge.

Previous Air Force work to simulate the missile endgame began with the FASTGEN program in the early 1970's. FASTGEN and and its file format were designed to calculate missile fragment trajectories, called shotlines, with intricate target models. The program keeps track of the target components each fragment intersects and the intersection geometry's details. This information is then used in a vulnerability program, such as COVART, to calculate a probability of hit for each target component.

The FASTGEN file format provides a powerful method for a designer to define an intricate target model by providing multiple base primitives. These primitives include a sphere, a cylinder, a donut, a box, a wedge and a rod in addition to the standard triangle (2:23). The FASTGEN file definition also allows the model's designer to define a thickness for both the model's interior and exterior components. Thus, for example, the pilot shielding on the target aircraft can be defined with a certain thickness. Using this factor, an endgame simulation program, such as SHAZAM, can calculate the component probability of penetrating the armor given the probability the armor was hit. By providing an accurate target model, FASTGEN provides accurate shotlines used to improve SHAZAM's probability of kill calculation for an endgame scenario.

After the shotlines are generated by FASTGEN, the component's probability of hit statistics are calculated by COVART. These calculations are used to determine the endgame's total probability of kill calculated by SHAZAM. COVART improves the probability of kill by modeling the components probability of hit to include certain additive effects. One such improvement is the capability to set target aircraft fuel levels (2:75). This capability allows the analyst to determine if the given target is able to complete its mission depending on the damage inflicted by the warhead's blast. For example, if the target's fuel tank is hit, it may affect the target's ability to finish its mission because it may no longer have sufficient fuel.

As described previously, SHAZAM is the main simulation program used to calculate the total probability of kill for an endgame scenario (3:6). The program and file format was developed so that multiple endgame trajectories could be handled relatively quickly compared to using the intricate FASTGEN models. The reason for this reduction in complexity was due to the lack of processor speed to calculate probability of kill using FASTGEN models.

Using target models usually converted from FASTGEN and reduced to triangle only models, SHAZAM produces an endgame's probability of kill by calculating each prioritized kill mechanism sequentially. SHAZAM's evaluation process includes the direct impact, warhead blast overpressure, and cumulative effect of warhead fragments interacting with the target's vulnerable parts (1:5). To calculate usable results, the program iterates a simulation a minimum of thirty times with the same encounter conditions to produce a single shot probability of kill that converges to the specified confidence level (1:5).

After calculating the probability of kill, SHAZAM uses a post-processor program to graphically display a wire-frame image of the exploding warhead and a blast fragmentation pattern expanding towards the target (1:10-11). Although SHAZAM is capable of producing accurate probability of kill results within a specified confidence interval, its graphical post-processor programs lack the realistic graphical appearance created by graphical libraries currently in use.

SHAZAM post-processors were the basis for other projects such as the Graphical Illustration of a Missile and Target Encounter (GIMATE). GIMATE illustrates specific SHAZAM endgame encounters, but it improved the graphical interface over the previous post-processors used by SHAZAM. One such improvement is a choice of either a black and white or color endgame illustration. This program was the next step in producing a more realistic endgame simulation given the limitations of processor speed and available memory. The GIMATE program showed that the capability to view the endgame encounter is invaluable for finding target model errors in the original SHAZAM simulations (3:6-7).

The most recent addition to software packages that simulate the missile endgame is the Ordnance Package Evaluation Code (OPEC). Like its predecessor SHAZAM, OPEC uses missile and target trajectory data, fuze sensor and target interaction data, and warhead and blast fragmentation data to calculate a target probability of kill. In keeping with the endgame simulation refinements over time, OPEC improves on SHAZAM in many ways. One improvement is that OPEC uses warhead fragment types of different shape, mass, and density. This provides a better mechanism used to calculate target damage done by those fragments. OPEC also allows the user to define the speed at which the blast fragments travel after the warhead's detonation (4:2-2). These additions add to the simulation's realism by improving the warhead fragment's behavior.

Although OPEC improves the methods used to determine the probability of kill of an endgame scenario, it does not generate scaled time movement of the simulation. Its only support of visual feedback is a static wire frame target with damaged areas indicated by a red asterisk. A major reason for the lack of graphical support in OPEC is the large number of polygons that make up the target files used by the program. The intricate models used to generate a better probability of kill are not conducive to producing moving graphical simulations.

To remedy the low resolution of OPEC in its graphical output, engineers at WL/MNMF implemented ENCOUNT. This program is designed to show various endgame parameters by taking a FASTGEN target and missile model, and combining them to display an endgame scenario. The resulting FASTGEN file can be viewed to see a static picture of the endgame when the missile is located at the miss point.

1.5. Assumptions.

The lack of scaled time simulation has prompted the production of a software system that supports this capability. However, before producing this software, I am making a few assumptions concerning its development. First, to create the endgame graphics, I am using the Inventor graphical libraries. Therefore, to remain compatible with the FASTGEN models as required by the statement of work, I must convert FASTGEN models into the Inventor format. As described later in chapter 3, I am using the OPEC GEORGE format as the basis for the conversion program. Therefore, I assume the programs necessary to convert FASTGEN files to GEORGE ASCII files are available. The current conversion process to a produce GEORGE ASCII file is shown in Figure 1-1. Boxes indicate file formats and arrows define programs that convert between formats.

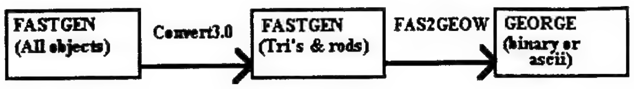


Figure 1.1: FASTGEN to GEORGE Conversion Process.

Second, I assume the availability of a Silicon Graphics computer running the IRIS operating system, version 5.3 or higher. The Inventor libraries are not compatible with previous versions of the operating system. Finally, I also assume the availability of the OPEC program to create the data files for use with the program.

1.6. Scope.

In my research, I will design and implement version 1.0 of the AFIT Missile Evaluation System (AMES). The areas I will focus on include

1. the basic graphics capability,
2. the program's input and output capability including the capability load missile and target endgame parameter information,
3. the capability to select and manipulate the target and missile models,
4. user modifiable dynamic vectors used to change the velocity and direction of both the missile and target,
5. simulation of multiple, transparent fixed cone fuze coverage patterns,
6. simulation of blast fragmentation and dispersion from a defined warhead origin,
7. the coloring of target components denoting the level of damage inflicted by the warhead blast, and
8. scaled time endgame entity motion.

Areas outside the scope of my thesis include

1. rendering the warhead detonation shock wave,
2. creating multiple level of detail target models,
3. providing a function for recording video of the simulation,

4. providing the capability to choose target, missile or world coordinate center point, and
5. providing coordinate feedback on the velocity vectors.

1.7. Approach.

The AMES project consists of six main areas. This section identifies these areas which constitute the bulk of the AMES project.

1.7.1. Conversion Software. The first step in producing AMES is to create a program that converts GEORGE formatted models into the Inventor format. To solve the problem of large complicated models, I will use a polygon decimation algorithm created by William Schroeder. When applied to a polygonal mesh file, the decimation algorithm reduces the total polygon count while keeping the shape of the model intact (5:65).

1.7.2. General Graphics Capability. The second step is to implement the capability to display the missile and target as interactive three dimensional models and the functionality to simulate the required endgame attributes. To accomplish this task, I will use the Inventor and Viewkit libraries developed by Silicon Graphics Inc. to create the interactive windows and pull-down menu bars used to provide the user with interactive control over AMES. The menu bar options will allow the user to change various simulation parameters according to the desired specifications. Changes made using the menus are reflected in the three dimensional view windows.

1.7.3. Dynamic velocity vector. Another requirement for AMES is for the user to be able to change the velocity and direction of either the missile or target through either direct manipulation of a velocity vector or through input data. This capability allows the engineer to create an endgame simulation with the missile and target moving in the desired direction and speed.

1.7.4. Dynamic movement. The principle goal of my research is to show the dynamic interaction of the missile's probability of kill determinants with the target. Thus, the user must have the capability to show missile and target motion. For example, when the simulation is started, the user can view the interaction of the fuze sensor coverage patterns as they pass through the target.

1.7.5. Blast Fragmentation. In my thesis work, I will generate the pattern created by an exploding warhead given an OPEC warhead data file. This OPEC file describes the speed and direction of the fragments defined for the given warhead.

1.7.6. Simulation control. Finally, my research will create the capability for the user to control the simulation by using a control panel. This panel will allow the user to play the simulation forward or backwards, and freeze the simulation to view a particular instant in time.

1.8. Required Materials.

To implement this software package, I require the appropriate FASTGEN missile and target models. In addition, I require the trajectory, burst point, point source and explicit warhead data files created by OPEC. These files describe the attributes of various endgame parameters that AMES will simulate. Finally, the project requires the Inventor, Viewkit, and Motif libraries to implement the graphical interface.

1.9. Thesis Presentation.

This thesis is divided into six chapters. Chapter II presents background information relevant to this thesis. In particular, it covers previous programs created to simulate and display graphical missile endgame images. Chapter III discusses the methodology I followed during AMES development. Included is a discussion of how the initial program requirements changed and how I adapted program development accordingly. Chapter IV described the design of the graphical user's interface and image conversion software. This discussion covers how I used Inventor to create the endgame entities and manipulate them according to the given endgame parameters. It also covers the use of Viewkit and Motif to create the necessary interface windows. Chapter V describes the project requirements that were satisfied and unsatisfied during my work. This chapter also defines how AMES can be used in the missile endgame simulation process. Finally, chapter VI recaps the efforts of my thesis and proposes recommendations for future work.

2. Background

2.1. Introduction.

In this literature review, I will answer the following questions:

1. What work has been done previously on air-to-air missile endgame simulation?
2. Why is there a need for the AFIT Missile Evaluation System (AMES) to graphically simulate the missile endgame?
3. What software library will I use to implement the missile endgame's graphical simulation?

Computer technology has developed at a phenomenal rate over the past thirty years with a new generation of technology being created about every eighteen months. Because of these advances, software developers struggle to keep pace and take advantage of these new technologies and ideas. Computer simulation is a particularly demanding arena attempting to benefit from these developments. Consequently, graphical missile endgame simulation has evolved, with existing programs being revised and new programs being developed using the advances in technology.

In this chapter, I first examine previous work done to simulate and graphically view the air-to-air missile endgame. Second, I examine how these programs have built on the successes of previous work and why there is a need for the AMES software project. Finally, I discuss a graphics software library that gives the software developer a powerful tool for producing graphical simulation programs faster and more efficiently.

2.2. Previous Work.

2.2.1. FASTGEN. Previous work done in conjunction with the Air Force to simulate the missile endgame began with FASTGEN. This simulation program was designed to calculate the intersection of parallel rays, called shotlines, through a target model. For each ray FASTGEN calculates, it keeps track of the target component it intersects and the details about the intersection geometry. This geometry data can then be used in a vulnerability program, such as COVART, to calculate probability of hit statistics for each target component.

To produce accurate ray intersections with a target, FASTGEN provides a method of defining a simulation model that gives the user multiple base primitives to build realistic model geometries and components. These primitives used to design a target model include a sphere, a cylinder, a donut, a box, a wedge and a rod as well as the standard triangle (2:23). The addition of these primitives gives the model creator more flexibility in replicating the specific details of a target thus improving simulation accuracy.

To improve model accuracy, the FASTGEN file format defines each model component with a unique four digit integer to distinguish it from other components. The target model creator assigns these numbers according to standard code categories that split the model's components depending on their type, either ground vehicles or aircraft (2:20-21). The creator uses these codes to label the specific target components, separating them from other model components. The model deaggregation makes it easier for the designer to create accurate models for use with the simulation software.

A model designer can define certain primitives, specifically the triangle, the sphere, the cylinder, the box and the wedge, in one of two modes. These modes are either plate or volume (2:23). Plate mode defines the primitives surface with a normal thickness such as modeling the surface of a fuel tank. Plate mode has the drawback that it can only define components with thickness between 0.01 inch and 2.99 inches (2:24). Volume mode specifies surfaces of thickness varying from a few inches to more than ten inches, such as the armor on a tank turret.

By defining the thickness of components, a probability of kill program (e.g. SHAZAM), can calculate the missile blast's effect on the armor by determining if fragments penetrate the component.

A requirement for the FASTGEN file format is that each component primitive must be associated with a unique single digit integer or space code. This code identifies the target region next to the primitive's surface. FASTGEN uses these codes to debug the target model and provides a means of altering the model's vulnerability depending on the space surrounding the component (2:21). Again, by improving the target model's accuracy, FASTGEN generates more accurate shotlines for use in a vulnerability calculation program.

There are multiple programs that a user can view a FASTGEN file to verify its correctness. One such program, INTRVUL, is capable of showing certain endgame parameters. However, INTRVUL has many limitations due to memory constraints and processor speed. One limitation is it can only show a maximum of 100 target components at a time (9:B-5). INTRVUL applies this limit because it calculates the probability functions associated with the kill mechanisms that FASTGEN uses while it creates the graphical images on the screen.

Another program a user can view a FASTGEN file is PLOT5. PLOT5 was written mainly to debug target models. However, it also provides a graphical representation of the interaction between the missile and target in the scenario calculated by FASTGEN (9:B-9). PLOT5 generates a two dimensional cross section of the target model and the damage inflicted by the warhead blast with the cross section's components.

IVAVIEW is another program that graphically displays a FASTGEN model. However, unlike previous viewing programs, IVAVIEW is capable of showing the entire model in three dimentions and not just certain parts or a cross section of the model. IVAVIEW renders the FASTGEN model from various elevations and azimuths for the user to verify model correctness.

It also gives the user the ability to shade the parts of the model to enhance visual feedback (10:1-4). However, the main advantage of IVAVIEW is it is designed for the Silicon Graphics hardware platform. Therefore, the rendering of the FASTGEN model is quicker, allowing the user to rotation and translate the model more efficiently.

2.2.2. SHAZAM. SHAZAM was designed to use vulnerability data output by COVART along with information on warhead and encounter conditions, to determine the probability of a missile warhead killing a target. It uses target models defined entirely of triangles to calculate multiple endgame trajectories relatively quickly. The program calculates sequentially the missile's prioritized kill mechanisms for the specific endgame encounter. These mechanisms include the missile's direct impact with the target, the warhead blast overpressure around the target, and the cumulative effect of warhead fragments interacting with the target's vulnerable parts (1:5). To achieve statistically significant results, SHAZAM computes values for these mechanisms a minimum of thirty times using different random numbers streams under the same encounter conditions. These iterations generate a single shot probability of kill that converges to a specified confidence level (1:5). By increasing the number of iterations on these calculations, the user can improve the single shot probability of kill statistic for the given encounter parameters.

Along with the multiple mechanisms used to calculate the probability of kill, SHAZAM developers created a file format for the target models used in the simultation. This format gives the user the power to define the size, shape and position of both objects in three dimensional space used to calculate multiple trajectories quickly. It also allows the developer to define the target's internally vulnerable components (1:2). The user can designate these components as being vulnerable to either direct hit and/or blast overpressure (1:7). SHAZAM uses these models as input for its simulations and determines the target's component interaction with the missile's kill mechanisms.

Usually, the SHAZAM model files are converted from previous FASTGEN models. However, the advantage of converted FASTGEN models is lost in the probability of kill calculation because SHAZAM uses probability of kill statistics derived from tables and not through direct calculation of fragment intersection with the target. The resultant probability of kill calculation thus loses accuracy.

For fuzing purposes, SHAZAM models the missile in the simulation as a point source. Within the missile, SHAZAM defines a fuze located at the missile's origin and consisting of a target detection component and a warhead trigger component. To model target detection by the missile, SHAZAM simulates a cone shape that extends beyond the missile to a preset maximum range. When the first point of the target intersects this cone, SHAZAM calculates an optimal time delay and then detonates the warhead with the fuze's trigger component. The time delay depends upon the simulation's encounter conditions including the velocity and direction of both the missile and the target (1:7,11). After warhead detonation, SHAZAM calculates the probability of kill with respect to the three types of kill mechanisms: direct hit, blast-overpressure and fragmentation (1:35). SHAZAM then combines the effect of these three calculations to get a single shot probability of kill for the endgame scenario (1:43).

After calculating the endgame simulation, SHAZAM can interface with the graphical software package such as FLYOUT to generate graphical images of a simulation. FLYOUT shows the approach and missile detonation of the scenario, with a blast fragmentation pattern expanding in a ring formation from the missile's detonation point (1:1, 10-11). To display the missile and target models, FLYOUT uses a simple wire frame representation. However, the problem with this graphical output is it does not convey to the user how the missile's kill mechanisms interacted with the target. It is difficult to see where the fragments intersected the target and very little can be acertained by how the blast overpressure effected the target.

Although SHAZAM is capable of producing accurate probability of kill results within a specified confidence interval, its post-processor programs, such as FLYOUT, lack the accuracy or realism to show the user the details of the SHAZAM simulation.

2.2.3. GIMATE. SHAZAM was the basis for the Graphical Illustration of a Missile and Target Encounter (GIMATE) project (3:6). GIMATE illustrates a specific missile endgame encounter as calculated by SHAZAM. This program was built to replace the wire frame based FLYOUT program and improves the graphical output of a scenario in main ways. One improvement over FLYOUT is a choice of having either a black and white wire frame simulation, a color wire frame simulation, or a full color simulation with the target's and missile's panels filled (3:7). In the panel fill mode, the GIMATE program gives color coded feedback as to the different target areas that are vulnerable to the missile's kill mechanisms. It uses the colors red, cyan, and yellow to describe invulnerable components, components vulnerable to direct hit or blast overpressure, and components vulnerable only to blast overpressure respectively (3:19).

Another option GIMATE gives the user is choosing what endgame components to show during the graphical simulation. These components include

1. target only,
2. missile and target,
3. missile, target and warhead fragments, or
4. missile, target, warhead fragments and target debris.

These choices give the user the flexibility to show only those entities needed for the particular simulation of interest (3:21-22).

GIMATE showed that the capability to view the endgame encounter is invaluable for finding target model errors in the original SHAZAM simulations (3:6). It was also instrumental in the production of the Advanced Medium Range Air-to-Air Missile (AMRAAM) system by providing illustrations of specific target, missile and fragment interaction of an endgame simulation. This information allowed engineers to debug their missile model and test it against simulated scenarios. During AMRAAM production, GIMATE also illustrated the potential contribution of missile debris to the lethality of a target (3:1). This program was the next step towards producing more realistic graphical feedback given the limitations of processor speed, graphics hardware, and available memory at the time.

2.2.4. OPEC. The most recent addition to software packages that simulate the missile endgame is the Ordnance Package Evaluation Code (OPEC). Like its predecessors, OPEC uses endgame trajectory data, sensor and target interaction data, warhead and blast fragmentation data as well as missile and target models for inputs to calculate the missile's probability of kill of the target. In keeping with the improvements over time in the missile endgame modeling, OPEC adds to the mechanisms used to calculate the endgames probability of kill. These improvements generate a better probability of kill statistic for a given scenario.

One improvement OPEC has made over SHAZAM is the capability of using warhead fragment types of different shape, mass, and density. OPEC also allows the user to define the missile fragment's speed after warhead detonation (4:2-2). These additions improve the realism and simulation accuracy by improving the warhead fragment's behavior.

OPEC also improved the simulation by incorporating an *airgap* data file into the calculation of the probability of kill. This file defines components vulnerable to fire and the air gap that exists between a component and its neighbor. When a missile fragment hits one of these components, OPEC increases the total probability of kill by a function of the airgap space. This function is designed to simulate a fire, causing the target's damage to increase (4:4-3). This capability adds to simulation realism and improves the probability of kill output by OPEC.

Due to the addition of these new capabilities and the subsequent increase in computational time, OPEC does not generate scaled time, moving images of the simulated engagement. Instead, OPEC only provides a static wire frame output of damaged target areas of the last trajectory data set run by the program. This lack of graphical feedback hinders the engineer using OPEC in viewing the missile's kill mechanism's interaction with the target.

A reason for the lack of graphical support is the large number of polygons that make up the target models used in OPEC. OPEC bases its file format on the FASTGEN format. However, instead of directly using the FASTGEN format, OPEC converts FASTGEN geometry files into a proprietary format called GEORGE. This conversion process approximates the original primitives in the FASTGEN file using triangles. The approximation algorithm greatly increases the number of total polygons in the resulting model. Also, since OPEC is designed for the Intel format, there is little graphical support to display three dimensional graphics images of the endgame encounter. The target model's intricate detail used to produce a better probability of kill is an Achilles' heel to OPEC's graphical feedback capability.

2.2.5. ENCOUNT. To remedy OPEC's low resolution graphical feedback, the engineer's at WL/MNMF developed ENCOUNT. This program was designed to take a FASTGEN target and missile model and combine them into a single FASTGEN file depicting an endgame scenario. Before combining the files however, ENCOUNT manipulates the target and missile according to the given endgame parameters. It also adds FASTGEN rod primitives and spheres depicting missile endgame parameters such as the relative trajectory and the relative miss point. The parameters included in the final FASTGEN file are listed below:

1. FASTGEN rod primitives depicting the target coordinate system without rotation.

2. FASTGEN rod primitives depicting the missile coordinate system rotated with the given elevation and azimuth.
3. The target rotated according to the given yaw, pitch and roll.
4. The missile rotated according to the angle of attack elevation and azimuth.
5. A sphere depicting the burst point (or relative miss point).
6. A sphere depicting the aim point.
7. A FASTGEN rod depicting both the missile's optimal and actual relative trajectory.

Using a FASTGEN viewing program IVAVIEW, the engineers can view the endgame file created by ENCOUNT (12).

Even though ENCOUNT supplements the graphical feedback of a particular endgame, there are still drawbacks to the program. First, it is only capable of showing one set of endgame parameters at a time. To view another set of endgame parameters, the user must rerun ENCOUNT with the new set of parameters and reload the FASTGEN file into IVAVIEW. This limitation causes delays in viewing multiple endgame parameters. A second drawback of ENCOUNT is it ignores feedback on fuze cone sensor coverage interaction with the target. It also lacks feedback on warhead fragmentation patterns as in OPEC. Finally, it does not show the target impact areas of the missile's fragmentation. An optimal program would combine the best features of the previously defined graphical programs so that a user could view all components of an endgame scenario.

2.3. *Method to Implement an Improved Graphical Air-to-Air Missile Endgame Simulation.*

The preceeding programs have improved air-to-air missile endgame simulation both in their probability of kill calculations and graphical feedback of a missile endgame. However, these programs do not take advantage of current graphics technology to produce a scaled time graphical picture of the missile's kill mechanisms interacting with the target. The AMES project plans to fill this void. AMES will provide the user an interactive, three dimensional graphics interface for viewing and studying the affects of a missile's kill mechanisms on a target.

To create this graphical interface, AMES will use the Open Inventor graphics library. This library is a set of objects and methods used for creating interactive three dimensional graphics applications (6:4). Open Inventor libraries take advantage of the powerful graphics hardware found on Silicon Graphics Onyx and High Impact machines and simplifies the programming effort for creating graphics programs. Included with the Open Inventor libraries are database primitives, interactive manipulators and three dimensional graphics components used to create interactive computer graphics. The various AMES components used to depicted an endgame scenario will be constructed from these objects.

Inventor takes advantage of object oriented programming by making every entity in the scene an object. This approach simplifies generation of graphical images by allowing the user to place any entities that appear in the scenario into a tree database structure. To render the scene, the tree database is applied to an Inventor viewer primitive. When the viewer is activated, Inventor traverses the scene graph and renders the objects in the database to the screen (6:5).

To simulate movement in an Inventor program, the library provides a primitive called an *engine*. This primitive supplies rate of change information to its connected transformation node causing Inventor to render effected objects in a different place in world coordinate space. These small changes simulate the object's movement. I plan to use this primitive to simulate the interaction between the missile's kill mechanisms with the target (6:12).

2.4. Conclusion.

In answering my first question, "What work has been done previously on air-to-air missile endgame simulation?", I have outlined the history of air-to-air missile endgame computer simulation, including programs for graphically depicting an endgame scenario. In summary, FASTGEN began the pursuit of air-to-air missile simulation by calculating fragment shotline intersections with a detailed target model. These shotlines were, in turn, used by COVART to calculate probability of hit statistics for the target's components. Finally, these statistics along with information on warhead and encounter conditions, were used by SHAZAM to calculate a total probability of kill for an endgame scenario.

The data created by SHAZAM could then be loaded into a graphical post-processor, such as FLYOUT, to visualize the missile's and target's interaction. However, FLYOUT's graphical output lacked accuracy and realism and demanded improvement. GIMATE improved upon FLYOUT by adding different viewing options for the user including a color and panel filled option. Furthermore, the color coded panels gave graphical feedback as to aircraft vulnerability in the simulation. Finally, OPEC built upon SHAZAM and its use of COVART and FASTGEN data and improved the calculations used for the probability of kill. However, OPEC adds no additional capability for graphically viewing the endgame scenario. This reduces the usefulness of OPEC's graphical feedback and prevents an engineer from viewing the missile kill mechanism's interaction with the target.

The second question answered was "Why is there a need for the AFIT Missile Evaluation System to graphically simulate the missile endgame?" Since past research has found that graphical feedback gives engineers better information as to simulation results, there is a demand for a scaled time graphical feedback program to test and view endgame scenarios. Therefore, the AMES project will produce this graphical program and fulfill all requirements for compatibility with OPEC. AMES will bring together the best attributes of all previously designed graphical post-processors using current technology.

The final question I answered was "What software library will I use to implement a graphical missile endgame simulation?" The answer to this question is the Inventor graphics library. This set of libraries gives me the power to produce a three dimensional graphics interface depicting an endgame scenario. It also has built-in primitives that simulate object motion. AMES will incorporate these primitives to simulate missile and target motion. By using the Inventor library, I will create a graphical simulation package that will visually depict all missile kill mechanism's interaction with a target.

3. Methodology

3.1. Introduction.

Previous missile endgame graphics post-processors have focused on creating a single frame of a simulation and not scaleable time graphics of endgame entity interaction. One reason for this focus was the lack of hardware and software capable of creating scaled time images. However, technology now exists that makes rendering scaled time images of a missile endgame possible. The main purpose of AMES is to use these new technologies to produce an interactive, scaled time, missile endgame simulation. To that end, this chapter outlines the choices I have made during AMES development.

First, I will cover two programs currently used by WL/MNMF to create and analyze endgame simulations. In this discussion, I will cover what each piece of software is capable of doing and how these capabilities shaped AMES requirements. Second, I will cover the initial AMES requirements as described in the statement of work. I include in this section the reasoning behind implementation decisions I made including which graphics library to use and what requirements should be emphasized during development. I conclude this section by describing changes made in the middle of production to the initial requirements and how I adapted AMES accordingly. Third, I discuss the requirement of making AMES compatible with the FASTGEN model file format and how I plan to fulfill this requirement given the constraints of a scaled time simulation. I finish this chapter by discussing both the satisfaction criteria AMES must meet and how I will measure them.

3.2. Missile End-Game Simulation.

3.2.1. Current Endgame Simulation. Past missile endgame simulation programs have focused on improving the underlying endgame model of previous simulation programs. In doing so, they have ignored implementation of scaled time graphic images. Instead, the only graphical feedback given to the user has been a static, single frame snapshot of the simulation showing the entities' positions at a given instant. An optimal program for an engineer at WL/MNMF would combine the graphical feedback of both ENCOUNT and OPEC as described in the previous chapters. It would also include feedback on fixed fuze cone interaction excluded by these programs. AMES is designed to fulfill these objectives.

3.2.2. Initial AMES Requirements. The initial requirement for AMES was to create a missile endgame simulation system that engineers could use to explain air-to-air fuzing concepts to personnel, experiment with sensor coverage concepts, and graphically demonstrate the results of fuze simulations. The program would be compatible with OPEC and use the trajectory, fuze and warhead data from this program to create a graphical simulation. It would also be compatible with the FASTGEN geometry file format and use FASTGEN missile and target files in its simulations.

The initial emphasis of the envisioned system was for the user to place a missile and target in a three dimensional space and orient the entities for the desired engagement using graphical manipulators. The user would also have the capability to move and stretch a velocity vector attached to each entity to supply directional movement information. Additionally, the user would have the capability to define one or more types of fixed or skewed cone fuze sensors to attach to the missile. The coverage pattern of these fuze sensors would be displayed depending on their attributes. The user would also have the capability to specify the type of warhead used on the missile. Depending on the data input, the warhead trajectory coverage would be displayed.

Finally, the user would have the capability to run the simulation at a selectable speed to show the fuze sensor and warhead interaction with the target. To this end, I began implementation of the basic AMES framework using these guidelines.

3.2.2.1. Inventor versus Performer. The initial crux of the project was to give the user as much manipulative capability over the simulation as possible. Therefore, I had to choose the graphics library to develop AMES based mainly on this requirement. My options were either the SGI Performer library or the SGI Inventor library. Since Inventor has a multitude of built in primitives that make it easy to select and manipulate an object in the scene database, I choose it as the basis for implementing AMES (6:4). Performer, on the other hand, has no built in primitives for manipulating objects within a scene. To implement movement of any kind within Performer, the user must create the code from scratch. This lack of built in primitives reinforced my decision to use Inventor.

To further explain the ease of manipulation in Inventor, I will describe a pre-defined Inventor manipulator primitive called a SoTransformBoxManip (8:676). When inserted into the Inventor scene database, the manipulator surrounds all objects to the right and below it in the tree with a group of Inventor draggers and manipulators. This set of manipulators allows the user to rotate or translate the object with six degrees of freedom.

Figure 3-1 and 3-2 show the capability of the SoTransformBoxManip. Figure 3-1 shows the manipulator surrounding a target model. After the SoTransformBoxManip surrounds the object, the user can click on any side of the box to translate the object in a two dimensional plane. This action creates the set of arrows on the side of the cube as shown in Figure 3-2. The SoTransformBoxManip also allows the user to rotate the selected object. Rotation can be accomplished by clicking on the edges of the box. After doing so, the user can rotate the object around the rotation axis that appears. Figure 3-3 shows this capability.

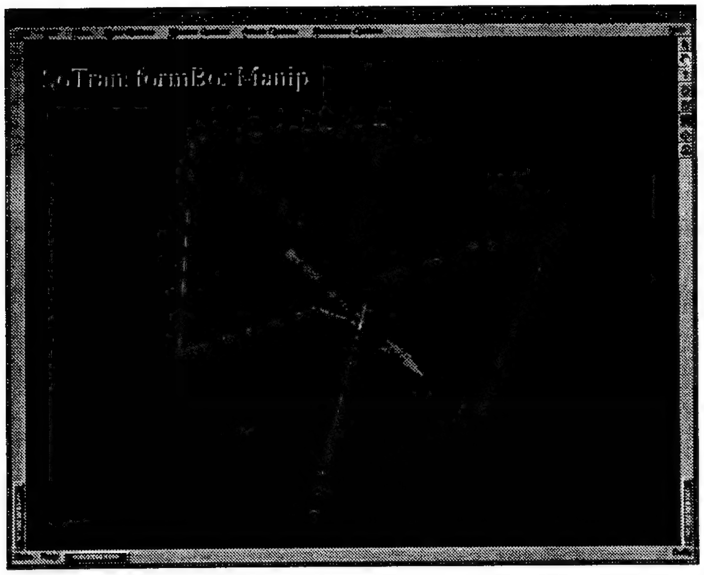


Figure 3-1: SoTransformBoxManip surrounding a target model

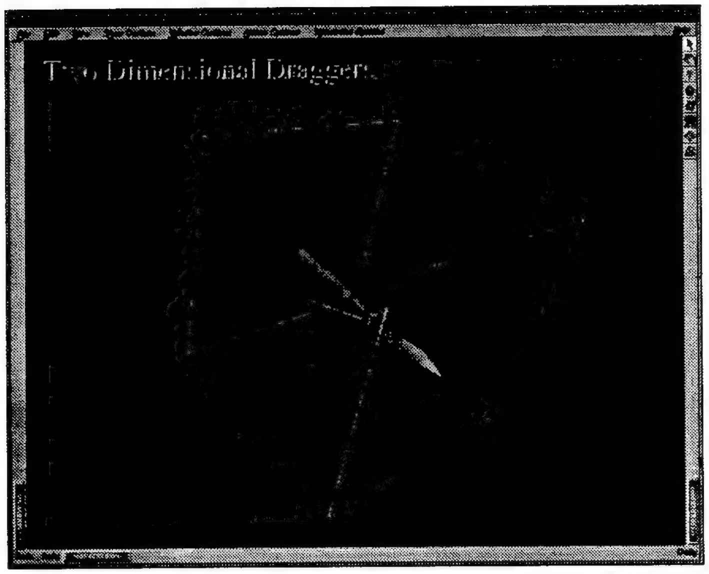


Figure 3-2: Draggers that appear after user clicks on the side of the box. The user can translate the box in the plane defined by the two arrows (Arrows have been bolded for better viewing).

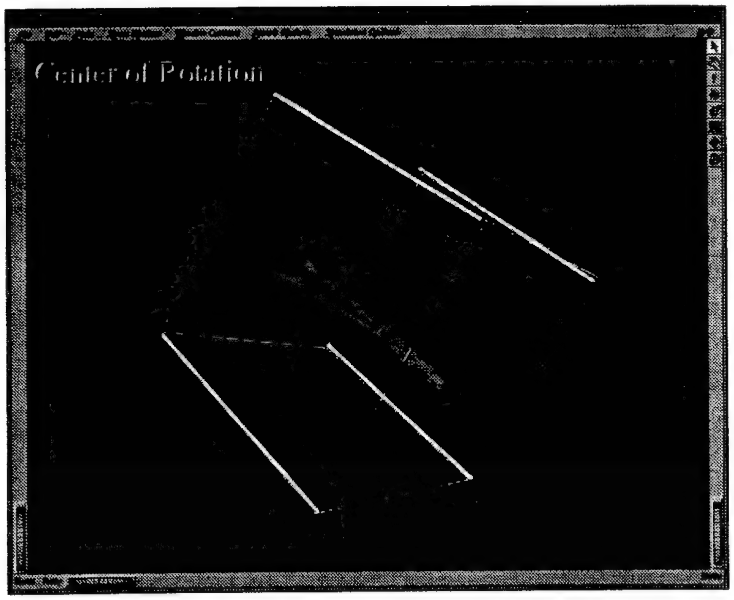


Figure 3-3: Shows the rotation capability of the SoTransformBoxManip. The rotation is accomplished around the center of rotation axis as shown. The four highlighted lines rotate around the center of rotation

. It is important to note that the SoTransformBoxManip node not only allows the user to translate and rotate the objects it surrounds but also to scale them. However, in term of AMES, the capability to scale the target's and missile's size is nonsensical. Therefore, I eliminated the scaling capability with this manipulator by writing a new default data file eliminating the scaling capability.

To define the capabilities of its manipulators, Inventor reads from a data file located in */usr/share/data/draggerDefaults*. First, to change the place Inventor looks for its manipulator data files, I defined an environment variable *SO_DRAGGER_DIR* to be the directory that the new default data file is located. The new file must be named with the same Inventor convension as the original data file so Inventor knows what to look for. In the data file, I eliminated only the scaling functionality. However, all other functionality remains according to the original manipulator definition. The data file acts like a Motif resource file by over writing the internally defined characteristics. Figure 3-4 shows the newly defined *transformBoxDragger.iv* file.

```
#Inventor V1.0 ascii

DEF transformBoxScalerScaler Separator {
}

DEF transformBoxScalerScalerActive SoSeparator {
}
```

Figure 3-4: transformBoxDragger.iv file

Another example of an Inventor manipulator primitive is the SoDragPointDragger. This dragger can be inserted into the scene graph and connected to a callback routine to translate an object or point in three dimensional space. Figures 3-5 and 3-6 show the capability of the SoDragPointDragger (8:221). The manipulator consists of two parts: a hexahedron and a cylinder. The hexahedron allows the user to move the manipulator in a two dimensional plane defined by the two long axes of the hexahedron. The cylinder allows the user to move the manipulator along the one dimensional line defined by the long axis of the cylinder. I am using this primitive to move and stretch the missile and target velocity vectors.

3.2.2.2. Graphical User's Interface Design. The envisioned graphical user's interface would allow the user to directly manipulate the missile and target in the endgame scenario using both mouse and keyboard input. The envisioned system would also allow the user to load data files from a menu interface that would manipulate the endgame according to the files data. With these requirements in mind, implementation of AMES began with the creation of the graphical user's interface using a GUI builder called Rapidapp (13).

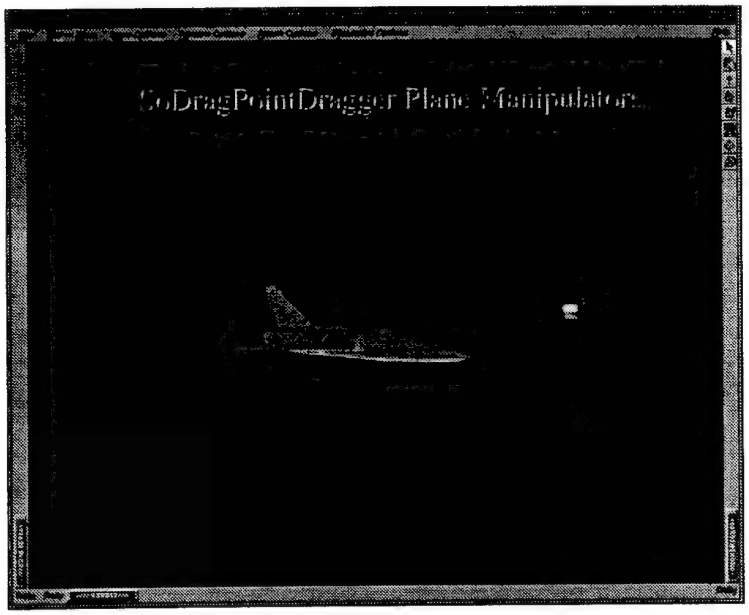


Figure 3-5: SoDragPointDragger plane manipulators. After clicking on the hexahedron section, the user can manipulate the velocity vector in the plane defined by the arrows.

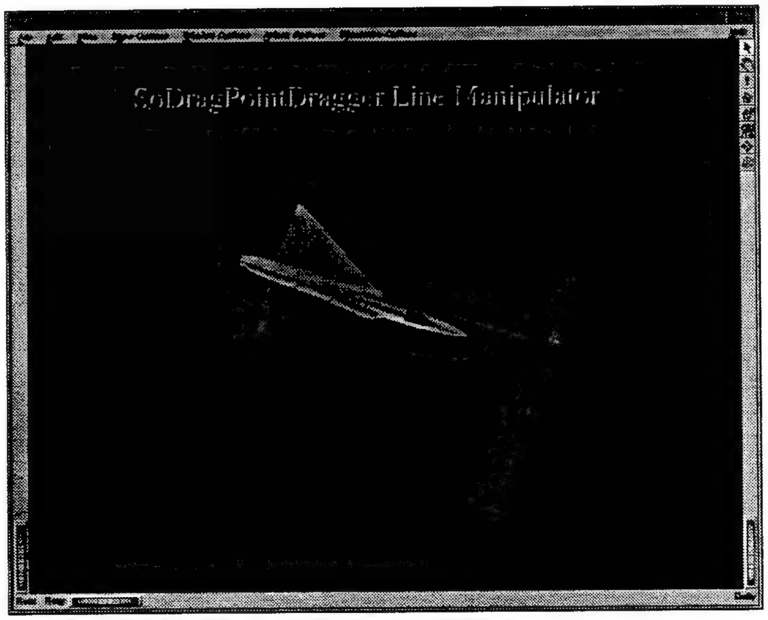


Figure 3-6: SoDragPointDragger Line Manipulator. After clicking on the cylinder section of the manipulator, the user can move it along the line defined by the arrow.

Although there are other GUI builders available such as FORMS, I choose Rapidapp because of it availability on the SGI systems in the graphics lab. I also chose Rapidapp because it gave me the functionality to create Motif windows and menus as well as incorporate Inventor view windows directly into the code. Since Inventor was chosen as the basis for AMES, this capability was an advantage. Another advantage of using Rapidapp was that the Motif windows and menus are built using the SGI Viewkit library. This library is an abstraction of the Motif library. It combines Motif commands to simplify the creation of windows and menus. By simplifying the interface with the X windowing system, system development time was decreased.

Rapidapp use was virtually seamless except for a few minor problems. First, the Makefile created by Rapidapp uses the make variable *(C++)*. To compile the code correctly, I had to change all references to this variable to *CC*, the C++ compiler on our SGI systems. Also, the Makefile assumed a compiler option of *-w3262* to shut off warnings about arguments which are declared but not referenced. I had to change this to *-w* for the *CC* compiler. Without these changes, the *make* program would crash when attempting to use the Rapidapp created Makefile.

During AMES development, the system administrators updated Rapidapp on the SGI systems to version 2.0. However, this new version is not compatible with previous versions of Rapidapp. Therefore, since I used Rapidapp version 1.1 to design the initial AMES graphical user's interface, I would recommend using this version for future additions to AMES to eliminate incompatibility problems.

Given the above listed requirements for AMES, the capability to create a dynamic simulation that the user can directly manipulate the missile and target was emphasized. Since I focused on creating this front end product including the graphical user interface, I began to incorporate plans to have AMES calculate the endgame parameters including the relative miss point and burst point for the simulation parameters. My plan was to incorporate OPEC algorithms into AMES to accomplish these calculations. I also began to pursue plans to simulate warhead fuzing. My idea was to use a collision detection algorithm to determine when a fuze cone had intersected a target. To fulfill the requirement of compatibility with OPEC, I also began work on calculating the OPEC trajectory and burst point data from the simulation created by the user. However, before these ideas were implemented, I presented an initial AMES prototype to WL/MNMF.

3.2.3. Requirement Clarification. After developing the initial AMES prototype, I delivered the system to WL/MNMF for requirement's clarification. During this trip however, the initial emphasis on creating a front end program changed. Instead of using the program to create endgame scenarios as emphasized in previous meetings, the engineers wanted a backend system that would graphically simulate a previously calculated endgame. They also expanded the compatibility requirements. In addition to being compatible with OPEC, the engineers requested AMES be compatible with ENCOUNT, the program written in house by WL/MNMF (12). They also requested I use ENCOUNT's view of the missile's azimuth and elevation rotations versus OPEC's. ENCOUNT and OPEC view these two parameters differently.

Adapting to the situation, I switched from creating an OPEC trajectory and burst file to using OPEC trajectory and burst file information to manipulate a target and missile to the specific endgame parameters. In doing so, I gave the user the capability to switch amongst multiple trajectory parameters included in the OPEC files without having to rerun the program as with ENCOUNT. I also created the capability to load an ENCOUNT data file and manipulate the target and missile according to the data. Although these changes comprised a major paradigm shift in AMES production, there were no major negative impacts on the development schedule or loss of earlier features. However, there was a need to make some initial requirements less important for implementation and more appropriate for future work. These requirements are described in Chapter 5.

3.2.4. Azimuth and Elevation difference between OPEC and ENCOUNT. Currently, OPEC views the missile's azimuth and elevation differently from ENCOUNT. The difference between ENCOUNT and OPEC's elevation parameter results from a difference in perspective. A positive elevation in ENCOUNT, the missile coming down on top of the target, is defined as a negative elevation in OPEC. The target is *below* the missile. Similarly, a negative elevation in ENCOUNT, the missile coming from underneath the target, is defined as a positive elevation in OPEC. The target is *above* the missile. The engineers want to use the ENCOUNT method for elevation because they view all endgame variables with respect to the target.

For azimuth, the degrees used to define a missile coming from in front and from behind the aircraft are reversed. In ENCOUNT, an azimuth of zero degrees has the missile coming directly towards the target from the front. The same attack position for OPEC is a missile azimuth of 180 degrees. The opposite is true for a missile coming from directly behind the aircraft. Figure 3-7 shows the differences between OPEC and ENCOUNT for both elevation and azimuth.

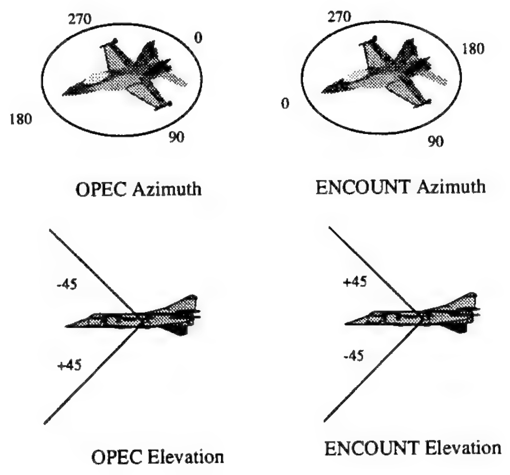


Figure 3-7: Difference of azimuth and elevation between OPEC and ENCOUNT

3.3. FASTGEN to Inventor Conversion Program.

One requirement of AMES is to make it compatible with the FASTGEN file format. Since AMES is built with Inventor, I need to convert FASTGEN model files to the Inventor file format. There are multiple options I could implement to meet this requirement. The following section describes these options and the one I chose.

3.3.1. File Conversion. One option to meet this requirement would be to directly convert a FASTGEN file to an Inventor file. This option, however, would require duplicating the triangle approximation done by the CONVERT program. Therefore, to avoid duplication, I decided against this option and used CONVERT to convert a FASTGEN file to triangles and rod primitives. The next option would be to convert the FASTGEN triangle and rod files to Inventor. However, since the requirement was to make AMES compatible with OPEC, I decided to use the *fast2geo* program distributed with OPEC to convert the FASTGEN triangle and rod files to the GEORGE file format. This eliminated the duplication of effort of approximating the FASTGEN rod primitives into triangles as done by *fast2geo*. Thus, I decided to use GEORGE files as the basis for conversion to Inventor.

One advantage of converting the GEORGE files to Inventor versus a direct conversion from FASTGEN is that it is trivial since the FASTGEN primitives have already been converted into triangles. A second advantage is simulation accuracy. The information used in AMES is OPEC created data. Since these data points are created using the GEORGE triangle model, the points simulated in AMES are identical and improve simulation realism.

Given this choice of conversion, I am faced with the increased number of polygons caused by the multiple conversion processes. To improve rendering of the large polygon count models, I am using polygon decimation to reduce the number of triangles in the model's mesh. By using this method however, I lose the advantage of simulating identical data points as described above.

However, this loss in model accuracy is acceptable considering the gain in motion rendering performance.

3.3.2. Polygon Decimation. One problem of using the FASTGEN format as the basis for the models uses in AMES, is that the large number of polygons in the intricate models is not conducive to motion rendering. The approximation of FASTGEN primitives into triangles during the conversion process also adds to the total number of polygons. As the number of polygons in the models increase, the processing time needed to render them substantially increases. This increase results in a noticeable slowdown of the rendering process especially when moving the model. This slowdown adversely effects the perception of continuous motion, producing poor animation effects. To remedy this problem, I implemented the polygon decimation algorithm by Shroeder to reduce the number of polygons in the target models (5:64).

Schroeder's decimation algorithm operates by making multiple passes over an existing triangle mesh using local geometry and topology characteristics to remove vertices that pass either a distance or angle criterion. If a vertex meets one of these criteria, the algorithm deletes it from the model and retriangulates the local mesh (5:65). By recursively using this process over the resulting mesh, the user can decimate the triangles until the desired, less detailed model, is produced.

Currently, I have written a program that reads in a GEORGE ASCII file, runs the Schroeder decimation algorithm on the polygonal mesh, and outputs a Designer's Workbench file (DWB). At the time this program was initially planned, DWB was the format I planned to use for AMES. In implementing the decimation algorithm though, I ignored deletion of what Schroeder calls boundary vertices. This decision was made to prevent introducing holes in the decimated model between the sections as defined in the GEORGE file. To complete the conversion to the Inventor format, the DWB file is loaded into the Designer's Workbench program and exported to an Inventor file (14).

The process of converting the GEORGE files to Inventor however, was improved by Capt. Jeff Bush. His program converts a GEORGE ASCII file directly to an Inventor file using the Schroeder Visualization Toolkit to reduce the number of polygons (19). In doing so, he eliminated the need for Designer's Workbench to complete the conversion to Inventor. The conversion program also adds a SoBaseColor node to each of the models objects. This addition simplifies the color coding of target components by AMES according to OPEC damage data (11:1-2).

With further research, I discovered that the designer's of OPEC have also written a program that converts a GEORGE ASCII file into VRML format. The VRML format is equivalent to the Inventor 1.0 format and can also be loaded by any Inventor viewer. However, their program, ACE2VRML, does not incorporate polygon decimation (15). Thus, the resulting VRML file has as many polygons as the original GEORGE file. As stated above, the large number of polygons is detrimental to scaled time simulation. Because of this result, I highly recommend further development of Capt. Bush's file conversion program.

3.4. Expected Results.

3.4.1. Satisfaction Criteria. AMES allows the user to display either previously generated missile endgame trajectory and burst point information or create a new scenario of a missile endgame. To consider AMES a success, it should display the simulation parameters according to the missile fuzing methods used by WL/MNMF. I am using ENCOUNT as the basis for correctness of endgame encounter parameters. I am using OPEC as the basis for correctness for the distribution of warhead fragmentation. All simulation parameters will also be verified by engineers at WL/MNMF.

3.4.2. Experiments. To test the satisfaction criteria, I will convert a FASTGEN missile and target file into Inventor files and load them into AMES. Next, I will calculate an OPEC trajectory and burst point file using the same two models. I will then load these files into AMES along with the same fuze and warhead information used to create the trajectory and burst data. Finally, I will select a trajectory and corresponding burst point from these files.

The first verification of correctness will be to compare the AMES graphical setup with an Ivaview setup of the same input parameters. Areas to be compared with Ivaview are: the target's yaw, pitch and roll, the missile's azimuth and elevation and the missile fuselage's angle of attack elevation and azimuth. Finally, the simulation burst point will be displayed in AMES and compared to the burst point in the equivalent Ivaview setup. I will use the expert opinion of WL/MNMF engineers to verify model correctness.

The second verification of correctness will include using OPEC to view the fragments distribution using the same input trajectory and burst data. The AMES simulation will be run and the warhead fragmentation output in AMES will be compared to the fragmentation in OPEC. Again, I will use the expert opinion of WL/MNMF engineers to verify model correctness.

3.5. Conclusion.

Although ENCOUNT provides feedback on parameters ignored by OPEC, it does not provide feedback on the fuze cones or warhead fragmentation of a simulation. On the other hand, OPEC displays the fragmentation pattern and target hit points of a missile simulation but ignores endgame information like the coordinate systems and relative trajectory between the missile and target. AMES will bridge the gap between these two programs giving fuzing engineers the capability of both programs.

AMES is designed to give the user feedback of fragment trajectory and target damage points as in OPEC as well as give the coordinate axes and relative trajectory feedback given in ENCOUNT. AMES also displays up to three pre-defined fuze cones and simulate the interaction of these cones with the target using scaled time simulation. By combining the parameters of OPEC and ENCOUNT and adding fixed fuze cone sensor coverage feedback and scaled time endgame simulation, AMES improves the capability of engineers to analyze missile endgame scenarios.

4. *Design and Implementation.*

4.1. Introduction.

This chapter covers AMES design and implementation. I begin this chapter in Section 4.2 with a description of the Inventor, Viewkit and Motif libraries I used to create AMES. In this section, I describe how adding nodes to the Inventor tree database effects the resulting scene and how I take these effects into account in forming the AMES Inventor tree database. I end this section with a description of the base Viewkit classes I use to create AMES' forms and windows. In Section 4.3, I describe AMES' overall system design. This discussion includes descriptions of all AMES' windows and forms as well as the program's capabilities. In Section 4.4, I describe the Inventor scene database I created for AMES. In this section, I first describe the tree's interior nodes, and then describing the tree's target and missile parts. I end this section with a description of the callback nodes I used to detect scene changes that initiate a response. Finally, in Section 4.5, I give a brief description of the endgame parameters including the AMES' fixed fuze cone parameters.

4.2. Graphical Interface Libraries.

To create AMES, I am using three graphics libraries: the Inventor library, the IRIS Viewkit library and the Motif library. The Inventor library provides the capability to create a three dimensional graphics window and place within the window the objects necessary to simulate the missile endgame. The IRIS Viewkit library provides the capability of creating window applications without having to deal with the underlying X windowing system. It includes capabilities to create an application menu and can easily integrate Inventor style widgets (16).

Finally, the Motif library is the basis for both Inventor and Viewkit and provides underlying functionality for creating the various windows needed for AMES. I am also using Motif directly to create the file dialogs used for file selection. These three libraries give me the power and flexibility to create the windows, menu items and graphical primitives necessary for AMES. The following is a description of each library.

4.2.1. Inventor. To create the endgame objects, the viewing window, and the endgame object manipulators, I used the Inventor library. This library is a collection of object oriented classes for shapes, objects, engines, and windows. It was designed for rapid development of an interactive graphics program. Thus, it is an excellent choice for building AMES. I am using instantiations of Inventor objects to simulate an endgame and so the user can directly manipulate the endgame parameters (6:4-6).

Inventor gives me substantial built-in computer graphics capability for user interaction. These capabilities include (but are not limited to) translating, rotating and scaling objects, interactive manipulators to move objects through the scene using the mouse, defining a scene view point via camera primitives, creating objects using polygon primitives and dividing or grouping scene parts with separator nodes. Using these nodes and capabilities, I built an Inventor tree database for the endgame scene (6:12).

Depending on how nodes are entered into the tree database, Inventor renders the scene differently. For example, to manipulate the missile's and target's yaw, pitch and roll, I am using the following Inventor nodes: a SoSeparator, a SoTransform and a SoInput. The SoSeparator node separates its children from other tree parts (6:49). The SoTransform node rotates, scales and translates all nodes that are to the right and below it in the tree, unless separated from the SoTransform node by a SoSeparator (6:42). The SoInput node loads in an Inventor input file. This node returns a SoSeparator node with children nodes representing the polygons or objects loaded from the Inventor file. I use this node to load into the scene graph the converted FASTGEN target and missile models (6:37).

In Figure 4-1, I have separated the missile model and it's transform node from the target model and it's transform node by a SoSeparator node. In this case, the missile and target are transformed according to their SoTransform node. Neither is affected by the others SoTransform node. However, in Figure 4-2, the missile's SoTransform node also transforms the target model. In this case, both the missile's SoTransform and the target's SoTransform nodes affect the target. The result is a combined scaling, rotation, and translation of both SoTransform nodes. Within AMES, the missile and target manipulation objects correspond to the design in Figure 4-1. By doing so, the missile and target are manipulated according to their own transformation matrices and are not effected by each other's transformation in the scene graph (6:46).

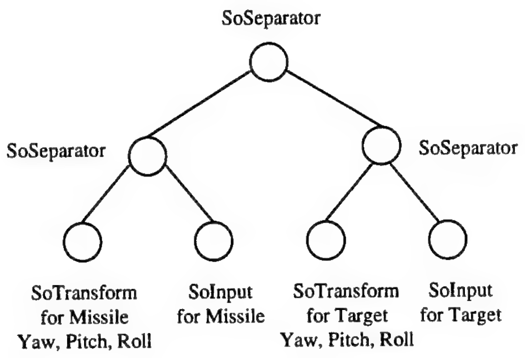

SoSeparator
SoSeparator
SoSeparator
SoTransform for Missile Yaw, Pitch, Roll
SoInput for Missile
SoTransform for Target Yaw, Pitch, Roll
SoInput for Target

Figure 4-1: Transformation and model nodes separated from each other. Models are effected by their transformation only.

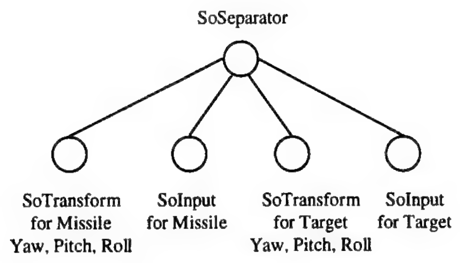

SoSeparator
SoTransform for Missile Yaw, Pitch, Roll
SoInput for Missile
SoTransform for Target Yaw, Pitch, Roll
SoInput for Target

Figure 4-2: Missile transformation node effects target model. Target model is translated according to it own transformation as well as the missiles.

Inventor camera nodes allow the user to view the scene database or watch the simulation unfold from any vantage point. The camera is applied to an Inventor view primitive and the scene is rendered from the camera's view point. Selecting the Inventor SoXtExaminerViewer primitive as AMES's main view window, I provide the user the built in thumbwheel controls to manipulate the camera's viewpoint. The thumb wheels rotate the camera position around the scene's focal point or cause the camera to zoom in and out from the scene. The user can also use the SoXtExaminerViewer to change the camera's position, rotate the camera around the focal point, and zoom in and out by directly manipulating the scene (8:723).

To simulate object movement, Inventor provides nodes called *engines*. I am using an Inventor engine called a SoElapsedTime engine to simulate each entity's movement (i.e., missile and target). For each entity, I attach an instantiation of the engine to a SoComposeVec3f node's X, Y and Z component. When the user activates the engines by pressing the control panel's play button, the engines feed translation data to the SoComposeVec3f's components. The addition of the three vectors, changes a transformation node and moves the effected object in the additive direction (8:229).

4.2.2. IRIS ViewKit and Motif Libraries. To implement the menus and AMES' main application windows, I am using the Silicon Graphics Viewkit library. This library simplifies the creation of these widgets by combining the underlying Motif functionality into a few steps. Like Inventor, Viewkit abstracts the method of window and menu creation, thus reducing coding time and improving application design. Although Viewkit simplifies the interfacing to the X windowing system by combining Motif function calls, I also make direct calls to Motif methods to create parts of AMES. For example, I directly instantiate a Motif *load file* window whenever the user selects a menu option for opening a specific data file. The following sections describe how I used Viewkit classes to create AMES.

4.2.2.1. Rumbaugh Diagram. Figure 4-3 shows a Rumbaugh inheritance hierarchy diagram of windows and forms created in AMES (17:38). All windows and forms shown in the diagram are either directly or indirectly VkComponent subclasses. In turn, all windows are subclasses of either the VkWindow class or the VkSimpleWindow class.

Figure 4-3 also shows the *burstPoint* and *encountData* class's inheritance from the *trajectory* class. These classes store information from either OPEC trajectory or burst point data files or an ENCOUNT data file, respectively. I used the *trajectory* class as the base class since it duplicated most of the variables found in the other two classes.

4.2.2.2. VkComponent. The VkComponent class is the base Viewkit class from which all subclasses are inherited. This class establishes the X server setup for each object, and provides methods to interact with the X window system. For example, all subclasses contain a method returning the Motif base widget for the class derived from VkComponent (16).

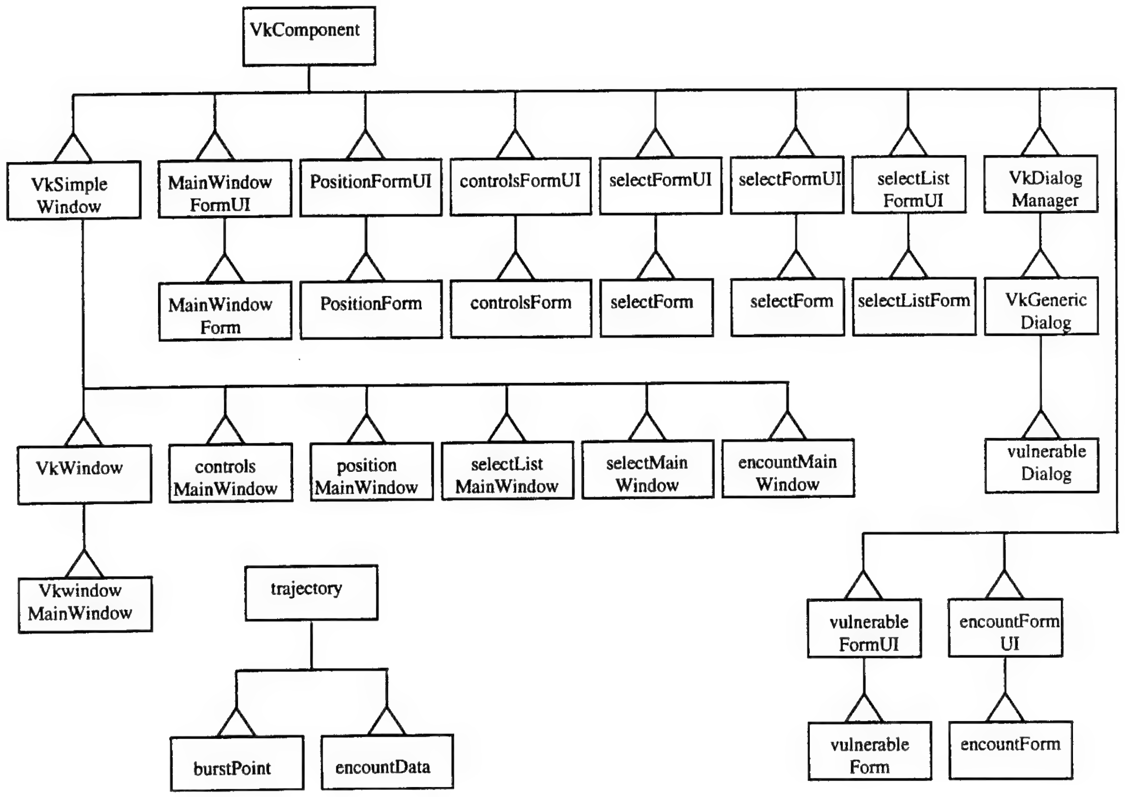


Figure 4-3: Rumbaugh diagram of inheritance hierarchy for AMES windows and forms.

4.2.2.3. VkWindow and VkSimpleWindow. The two classes VkWindow and VkSimple-Window inherit from VkComponent and are designed for use as base windows in Viewkit applications. All top level windows must be instances or subclasses of one of these two classes. The difference between the two classes is that VkSimpleWindow has no built in menu support while VkWindow, which is also a subclass of VkSimpleWindow, has menu support. Because of this difference, I have used the VkSimpleWindow class to create objects such as the AMES control window and the AMES position and velocity feedback window for which no menus are required. Similarly, I used VkWindow to create my main application window class VkwindowMainWindow, because it required menu support. Both of these window classes work with the main application window VkApp to provide support for mouse interaction with the windows (e.g. setting cursors in text fields and entering busy states) (16).

4.2.2.4. VkGenericDialog. VkGenericDialog is an abstract class that defines a specific dialog box without having to instantiate the dialog at run time. Instead, AMES creates the dialog only at the moment the user needs it. Then, when the dialog is unmanaged, AMES deletes it from memory, saving space. I initially used this class to create the vulnerable camera point selection dialog. However, to provide better user feedback, I also use pre-defined Viewkit subclasses of VkGenericDialog. These dialog subclasses include VkInfoDialog and VkErrorDialog. Each dialog is designed to give different feedback appropriate to the situation met by the user (16).

4.3. Overall System Design.

I have designed and implemented AMES to satisfy three main objectives:

1. Create an interactive graphical interface for endgame encounters, including target and missile placement and movement.
2. Simulate endgame object interaction, including warhead fragment trajectories and fixed fuze cone coverage patterns.
3. Adhere to compatibility with OPEC and ENCOUNT data files.

The following is a description of AMES functionality implemented to satisfy these main objectives.

4.3.1. Graphical Interface Components and Functions. Jakob Nielson defines usability of an interface as having multiple components and traditionally associated with the following five attributes:

1. *Learnability:* The system should be easy to learn so the user can start getting work done immediately.

2. *Efficiency*: The system should be efficient to use, so that high productivity is possible.
3. *Memorability*: The system should be easy to remember, so the casual user can return to the system after a period of time and begin using it without having to relearn the interface.
4. *Errors*: The system should have a low error rate, so that users make few errors while using the system.
5. *Satisfaction*: The system should be pleasant to use. (18:26)

I took these five points into account while designing the AMES graphical user interface. The following is a description of the components I built for AMES and how I used Nielson's usability list during development.

4.3.1.1. View Windows.

4.3.1.1.1. Main View Window. AMES main view window is derived from the Inventor SoXtExaminerViewer class. This viewer, as described previously, gives the user multiple capabilities for viewing and manipulating the scene. The following list outlines the main built-in features supplied by the viewer:

1. The viewer has three thumb wheel components. Two thumb wheels allow the user to rotate the camera around the camera's focal point either from the scene's top to bottom or side to side. The third thumb wheel allows the user to zoom in or out from the scene.
2. The viewer's camera point of view can be changed by the user by directing manipulating the scene itself. Using various combinations of mouse keys, the user can duplicate the camera's rotation around the scene as provided by the thumb wheel's. However, by manipulating the camera with the mouse, the user can also move the camera around the scene in any direction. Finally, the user can also translate the camera up and down or side to side in the scene.
3. The viewer has a built in pop-up menu that provides multiple options for the draw style of scene objects. Among these options, is the capability to draw the objects in wire frame or hidden line as well as a closed polygonal object. This capability is to AMES

advantage because the user can view the aircraft's interior using these draw styles to view color coded damage feedback.

For a complete list of all the SoXtExaminerViewer capability, reference the Open Inventor C++ Reference Manual (8:723).

The camera's manipulative capability described up to this point includes the camera's position being moved around a specific focal point. However, another AMES requirement is to allow the user to choose a target vulnerable point and view the simulation in any direction from this point. To accomplish this requirement requires an extension to the SoXtExaminerViewer's camera manipulation capability. Therefore, I added three additional thumb wheel components to AMES derived SoXtExaminerViewer class to accomplish camera rotation without changing the camera's position.

To implement camera rotation without moving the camera's position, I assume a coordinate system with the camera's focal point down the negative Z axis of a right handed coordinate system. I define each thumb wheel's rotation according to this coordinate system. Thus, a rotation around the X axis looks to the user like the camera is pitching up or down. A rotation around the Y axis looks to the user like the camera pivoting from side to side. Finally, a rotation around the Z axis looks to the user like the camera is rolling. Figure 4-4 depicts the coordinate system and the camera's focal point rotations.

An additional function added to the main view window is a thumbwheel to increase and decrease the viewing frustum's height angle. By rotating the wheel towards the viewer's top, the height angle decreases. By rotating the wheel towards the viewer's bottom, the height angle increases. This capability allows the user to view more of the scene from a specific view point. An additional capability of this thumb wheel though, is a button located just beneath the wheel.

This button, when pressed, resets the viewing frustum's height angle to its default value. Figure 4-5 shows the location of this additional thumb wheel.

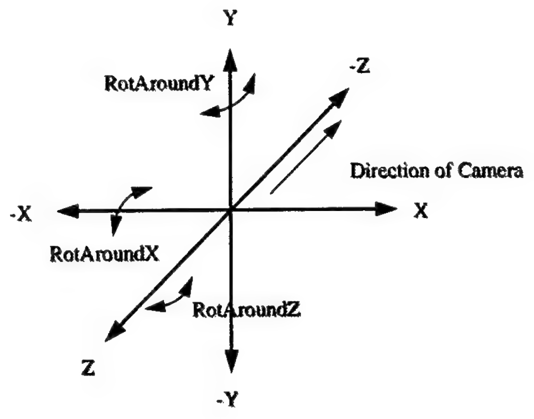


Figure 4-4: Description of the Camera coordinate system and the thumb wheels that rotate the camera's direction

4.3.1.1.2. Three Position View Windows. Built into AMES are three additional SoXtExaminerViewers that initially view the simulation from set points on the world coordinate X, Y, and Z axis. The user can view the simulation from four points simultaneously, thus providing improved user feedback. Since these windows are instantiated objects of the SoXtExaminerViewerDerived class like the main window, the user has the same manipulative capability in each of these windows as the main window. However, since these windows are smaller, the additional camera rotation thumb wheels are not included. To create the maximum viewing area, AMES initially turns off the decoration around each window. The user has the option of turning the decoration on by selecting the option on the viewer's pop-up menu. Figure 4-5 shows the three view windows activated.

4.3.1.2. Loading Models or Data files into AMES. The user can load multiple data files

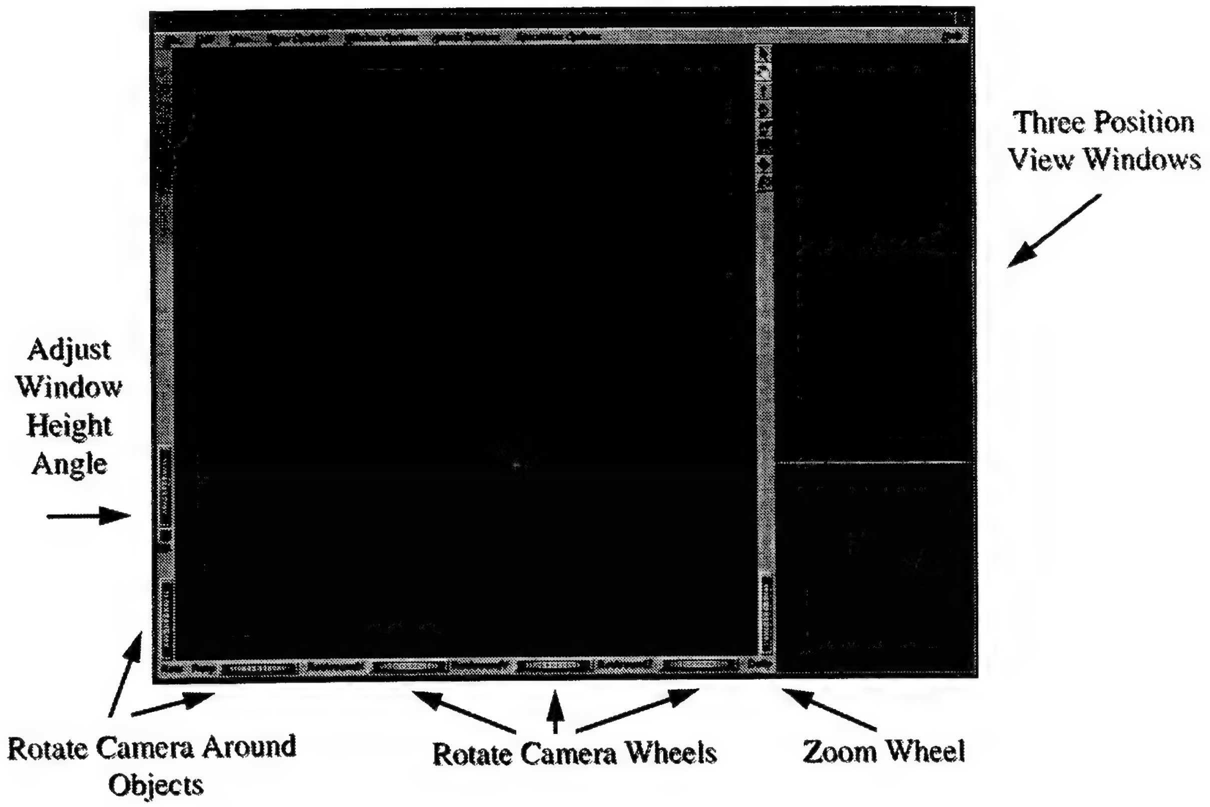

Three Position View Windows
Adjust Window Height Angle
Rotate Camera Around Objects
Rotate Camera Wheels
Zoom Wheel

Figure 4-5: AMES's Main Window with Three Position View Windows activated.

into AMES to create an endgame scenario. However, to display an accurate simulation, the loaded data files and models should be consistant. If not, AMES assumes the information to be correct and attempts to apply trajectory and target damage information to the target and missile models loaded. AMES can load the following endgame data files:

1. Inventor target and missile models,
2. OPEC trajectory and burst data,
3. ENCOUNT data,
4. OPEC Output data,
5. OPEC point source and explicit warhead data, and
6. AMES fixed fuze cone data.

Figure 4-6 shows the Motif window activated for the user to select an input file.

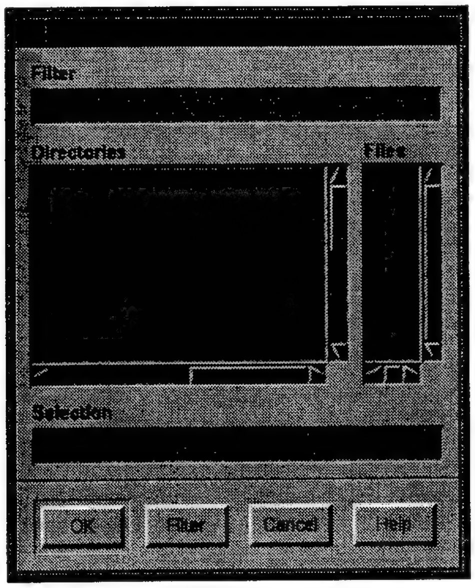


Figure 4-6: AMES window used to load all input files.

4.3.1.3. Edit Menu.

4.3.1.3.1. Edit Background Color. The user has the option to change the view window's background color. Depending on the use of AMES, changing the background color may be important to convey the correct endgame situation. For example, there may be a requirement to show a simulation during the night versus the day to emphasize a presentation. Another use of changing the background color is to accentuate a specific damaged area. Figure 4-7 shows the pre-defined Motif window used for this function.

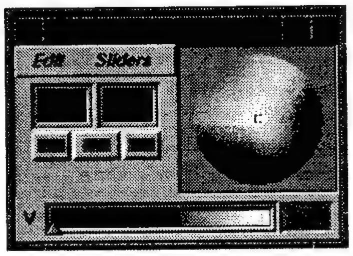


Figure 4-7: AMES window used to change the background color.

4.3.1.4. View Menu.

4.3.1.4.1. Simulation Control. AMES has a simulation control window that consists of a play, rewind, fast forward, pause, stop and record button. The user can use these buttons to control the simulation state. Keeping in mind Nielson's need for ease of learnability, the buttons act in the same manner as a video cassette recorder. The following is a description of each button's function:

1. The play button starts the simulation. When pressed, the missile and target move in the direction and speed provided by the velocity vectors.
2. The fast forward button increases the simulation speed by a factor of two.
3. The rewind button reverses the simulation at the same speed as the initial velocity vectors.
4. The pause button freezes the simulation in its current position. When the user clicks the pause button a second time, the simulation continues from the paused position.
5. The stop button causes the simulation to freeze in time. When the user hits the play button again, the simulation continues from the point the user previously stopped it.
6. The record button will start the simulation but also saves the initial simulation conditions to a file. Later, the user can read in the saved file later and duplicate the same simulation. Currently, this button is not implemented.
7. The close button unmanages the widget until selected again from its menu option.

Figure 4-8 shows the control window.

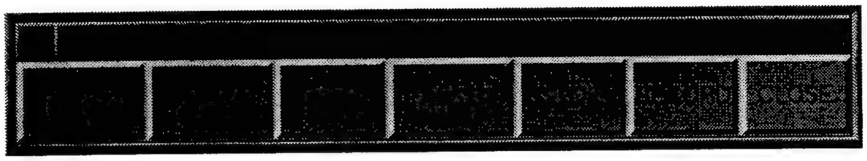


Figure 4-8: AMES control window.

4.3.1.4.2. Missile and Target Feedback Window. This window provides feedback of the current X, Y and Z position and velocity magnitude of either the target or missile. AMES updates these values when the simulation runs or when a user manipulates the objects directly. The user can also change these values using the keyboard. After hitting the enter key, the changes made by the user take effect. It is important to note that a change in the velocity magnitude will extend or contract the current velocity vector. However, the direction remains the same. The close button unmanages the widget until selected again from its menu option. Currently, the points shown in this window are in world coordinates and the units are meters. Figure 4-9 shows this window.

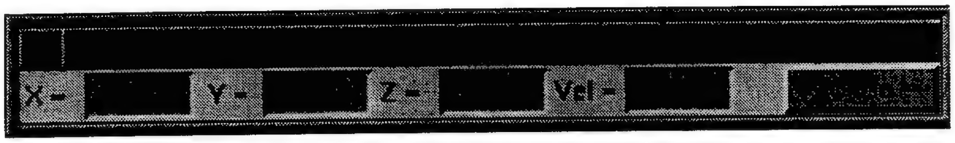


Figure 4-9: Missile and Target Position and Velocity feedback window.

4.3.1.5. Show Options Menu. The options in this menu toggle the visibility of the following AMES features:

1. the missile and target velocity vectors,
2. the three position view windows,
3. the missile's fuze cones,
4. the warhead fragment trajectories,
5. the relative trajectories, and
6. the target's and missile's coordinate system axis.

These toggle options serve two purposes. First, they help to enhance performance by removing objects not needed for the specific simulation. For example, when the three position view windows are active, they greatly decrease render speed. Therefore, it is practical to use these to view a static endgame picture versus watching endgame entities interacting with each other. Second, the toggle options give the user the flexibility to create a simulation according to particular requirements. For example, a fuzing engineer can eliminate the coordinate axis and relative trajectories and only view the missile's fuze cones and fragment trajectories.

4.3.1.5.1. Velocity Vectors. Both the missile and the target have attached velocity vectors the user can directly manipulate. Attached to one end of the vector is the Inventor SoDragPointDragger manipulator. By moving this manipulator, the user can change the direction and magnitude of the model velocity vector. As the user moves the manipulator, AMES also displays the current vector magnitude in the position window as described above.

AMES also transforms the velocity vector when the user loads and selects either OPEC or ENCOUNT trajectory data. However, it is important to note that the user has ultimate control over AMES simulation parameters and can still manipulate the velocity vector and other parameters after they has been changed according to the selected data.

4.3.1.5.2. Relative Trajectories. AMES calculates two relative trajectories. The first is the optimal relative trajectory. This trajectory is depicted as a yellow Inventor line that goes through the given target aim point. The other trajectory line is the actual relative trajectory. This trajectory is depicted as a light blue line that goes through the given relative miss point. AMES duplicates the calculations for these trajectories using the ENCOUNT code provided by WL/MNMF (12:1).

4.3.1.5.3. Target Damage Feedback. AMES is capable of showing the target damage calculated by OPEC. AMES color codes the damaged target components depending on the average number of hits in the OPEC output file. When the user selects the target damage option, AMES applies the appropriate color to each object. Table 4.1 shows the RGB values used depending on the components average number of hits.

Number of Hits	RGB Values		
0 < hits <=10	0.0	0.0	1.0
10 < hits <= 20	0.0	0.5	1.0
20 < hits <= 30	0.0	1.0	1.0
30 < hits <= 40	0.0	1.0	0.5
40 < hits <= 50	0.5	1.0	0.0
50 < hits <=60	1.0	1.0	0.0
60 < hits <=70	1.0	0.85	0.0
70 < hits <=80	1.0	0.5	0.0
80 < hits <= 90	1.0	0.25	0.0
90 < hits <= 100	1.0	0.1	0.0
hits > 100	1.0	0.0	0.0

Table 4-1: Definition of RGB color values given to objects depending on hit damage recorded in OPEC output file.

Table 4-2 shows an example of the OPEC output file used to determine the color code for a damaged object. The only data point AMES currently uses is the average number of hits. AMES loads all other data points into a *hitDamage* class object for future use. It is important to note that when a second set of target damage data is loaded, the original data is deleted and the target damage toggle is turned off. To view the new target damage data, the user must reselect the target damage menu item.

Object Name	Avg # Pk	Avg # Hits	Avg Impact Hits	Avg Impact Mass	Avg Initial Speed	Avg Initial Frag. Vel.
...						
3430_o4	1	1	0	449.996	811.967	4966.925
3430_o5	1	3	0.02	588.936	4140.109	5654.896
3430_o6	1	2	0.01	494.962	4052.688	5401.567
3430_o7	1	1	0	723	5376.738	5376.738
3430_o8.r1	0	0	0	0	0	0
3430_o8.r2	0	0	0	0	0	0
3430_o9.r1	0	0	0	0	0	0
3430_o9.r2	0	0	0	0	0	0
3441_o0.r1	0	0	0	0	0	0
3441_o0.r2	1	1	0	401.497	4520.127	5351.748
3441_o0.r3	0	0	0	0	0	0
4001_o0	0.108	536	2.68	565.742	2587.387	5297.348
4002_o0	0.073	440	2.2	566.766	2257.347	5333.24
...						

Table 4-2: Table of OPEC output data. This data is used to color the vulnerable points according to the average number of hits.

4.3.1.5.4. Fragment Trajectories. AMES displays the fragment trajectories of a given point source or explicit warhead data file as Inventor lines emanating from the warhead origin point. The fragment lines are calculated by vector addition of the fragments speed and trajectory with the missile's current velocity vector. The lines are colored in a graduated scale from red to yellow to give depth to the coverage pattern. For the point source warhead, the lines are also colored to depict from which polar region the fragments emanate.

The explicit warhead fragments are drawn according to the information contained in the data file. However, for the point source warhead, the warhead trajectories are calculated by making two random draws between zero and one from a multiplictive congruential random number generator. These random draws determine the actual polar and azimuthal angle the fragment line follows. The polar angle is a scale factor of the first random number between the minimum and maximum polar angles defined in the OPEC file. The azimuthal angle is a scale factor between zero and 360 of the second random draw. With these two angles, the fragment line is calculated using equation (1). Figure 4-10 depicts the two angles.

$$S = s \cos\phi \; i + s \; \sin\phi \cos\theta \; j + s \sin\phi \cos\theta \; k \qquad (1)$$

In equation (1), *s* is the vector component scaling factor, ϕ is the polar angle and θ is the azimuthal angle. This calculation is done for each of the fragments defined in a polar region.

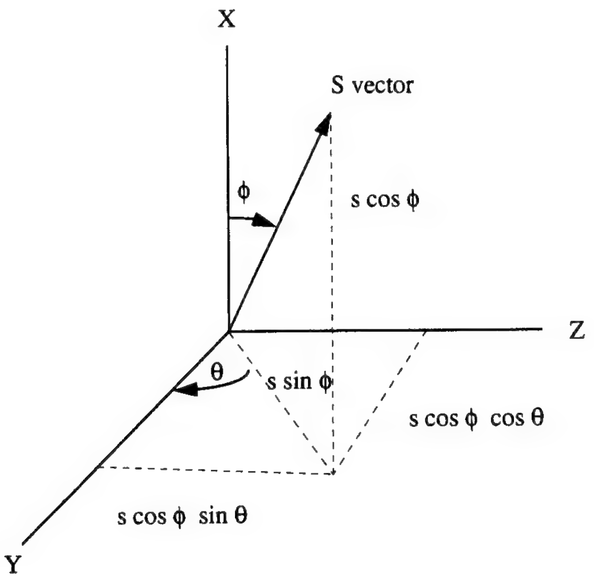


Figure 4-10: Point Source Warhead calculations.

4.3.1.5.5. Coordinate Axes. AMES calculates the coordinate axes for both the missile and target depending on the rotations given by the endgame parameters. The axes are defined as three Inventor lines depicting a right hand coordinate system. It is important to note that the target's coordinate axis are not transformed by any endgame parameters. Initially, they are equivalent to AMES world coordinate system. However, if the user selects and manipulates the target, the coordinate axis will move with it. The coordinate axes also move with the target during scaled time simulation.

The missile's coordinate axes are transformed according to the missile's azimuth and elevation given in the endgame parameters. Like the target coordinate axes, they also move with the missile if it selected and manipulated by the user. Finally, they also moves with the missile when the user starts the simulation.

4.3.1.6. Window Options Menu.

4.3.1.6.1. View All Option. AMES provides a *view all* option that centers the four view window cameras on the scene objects. It is provided for use when the simulation objects proceed beyond the confines of the camera's current view frustum. To provide this functionally, AMES uses a built in function provided by the SoXtExaminerViewer called viewAll().

4.3.1.6.2. Reset Scene Option. The reset scene option places the missile and target back to the world coordinate origin. This option eliminates all rotations and translations applied to each object. The purpose of this option is to allow the user to start from scratch in creating a new simulation.

4.3.1.7. Attach Options Menu.

4.3.1.7.1. Missile and Target Camera Viewpoints. The user can choose amongst multiple view points, apart from the initial main and three position window viewpoints. Included in these choices is the option to view the simulation by tethering to either the missile or target. When the user selects either of these choices, the main viewpoint attaches behind the chosen object. When the simulation is run, the camera follows the attached object.

4.3.1.7.2. Attach Vulnerable Point. The user can choose a point in the target coordinate system to view the simulation (typically a most vulnerable point.) This camera acts in the same manner as the attached missile and target cameras. When the simulation is run, the camera stays attached to the target. Figure 4-11 shows the window to define the vulnerable viewpoint.

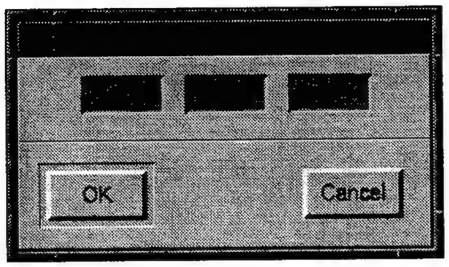


Figure 4-11: Select vulnerable camera point window. The point is in target coordinates.

4.3.1.8. Simulation Options.

4.3.1.8.1. Selecting OPEC Trajectory Data. The user can choose from any trajectory in the loaded OPEC trajectory file. There are two ways the user can accomplish this task. First, the user can type the number of the trajectory into the text field and press enter (AMES assumes the first trajectory in the file as number one). This method causes the missile and target to be manipulated according to the endgame parameters without closing the selection window. The second method is to type the trajectory number into the text field and click on the *Ok* button. This method causes the missile and target to be manipulated and closes the selection window. The window's *Cancel* button closes the window without effecting the scene. The selection window to choose an OPEC trajectory is shown in Figure 4-12.

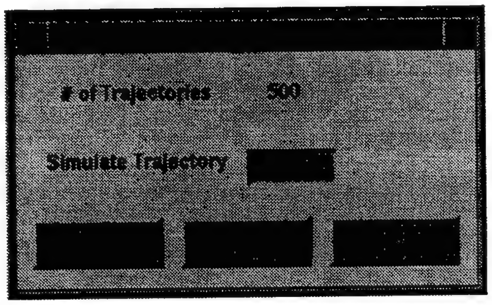


Figure 4-12: OPEC trajectory selection window. After entering the number, the selected trajectory data is displayed in graphical form.

4.3.1.8.2. Viewing OPEC Trajectory and Burst Point Data. Additionally, the user can view the current OPEC trajectory and burst point data by clicking on the *List* button on the trajectory selection window (See Figure 4-13). The user can scroll through the trajectory and burst point information currently loaded using this window.

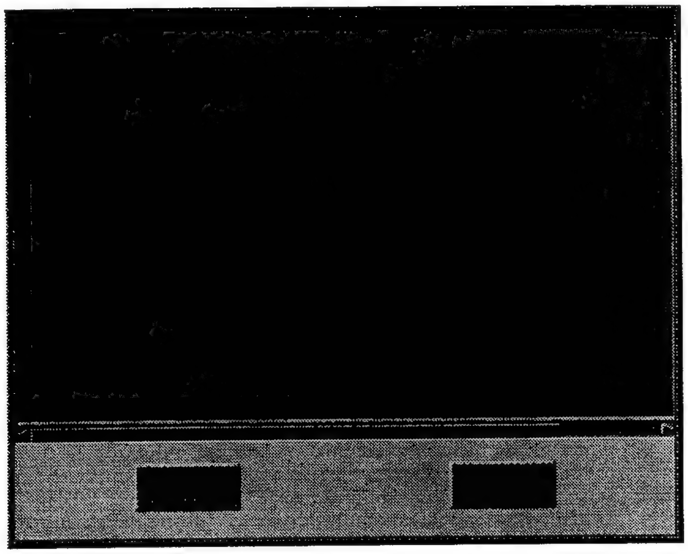


Figure 4-13: AMES window to view the trajectory and burst point data currently loaded.

4.3.1.8.3. Selecting ENCOUNT Data. AMES has the dual capability of showing both OPEC and ENCOUNT endgame trajectory data. The dialog window for viewing and modifying ENCOUNT data is shown in Figure 4-14. AMES gives the user two options for entering ENCOUNT data. First, the user can load the data from an ENCOUNT input data file. After a file is loaded, AMES shows the information in the ENCOUNT selection window. The second option for simulating ENCOUNT data is to enter the data into each ENCOUNT text field. Since each text field is editable, the user can change each parameter separately depending on the endgame parameters desired. When the user clicks on the *Ok* button, AMES manipulates the target and missile endgame parameters to the values currently shown in the ENCOUNT window. and closes the ENCOUNT selection window. However, the user can change the endgame parameters without closing the ENCOUNT window by pressing the enter key with the cursor located in one of the window's text fields. This form was created by Capt. Jeff Bush using his reusable Motif code written during his previous assignment. I then added the callback capabilities needed to apply the entered data to the endgame scenario.

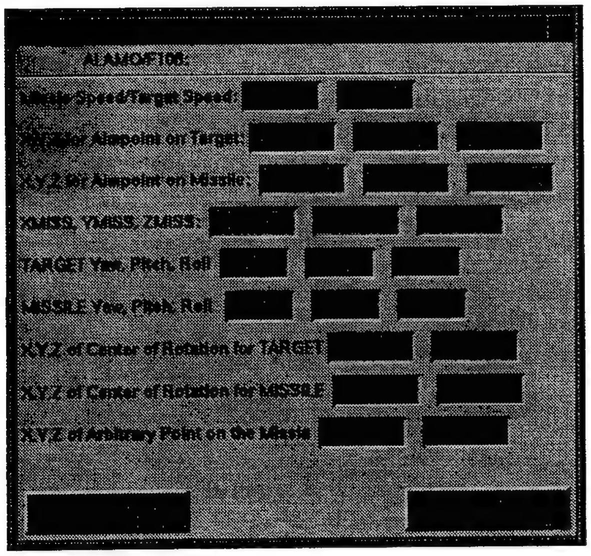

ALAMO/F100:
Missile Speed/Target Speed:
X,Y,Z of Aimpoint on Target:
X,Y,Z for Aimpoint on Missile:
XMISS, YMISS, ZMISS:
TARGET Yaw, Pitch, Roll
MISSILE Yaw, Pitch, Roll
X,Y,Z of Center of Rotation for TARGET
X,Y,Z of Center of Rotation for MISSILE
X,Y,Z of Arbitrary Point on the Missile

Figure 4-14: AMES window to display and change ENCOUNT parameters.

4.3.1.9. Error Dialog Boxes. To provide the user with appropriate feedback as to AMES' state, I built error dialog boxes using pre-defined Viewkit dialog types. The advantage of using this method was that all the dialog boxes are created dynamically versus statically. In other words, a dialog box is only created when needed and does not exist in memory throughout the use of AMES. It is subsequently deleted from memory when the user clicks the *Ok* button. Figure 4-15 shows an example dialog box.

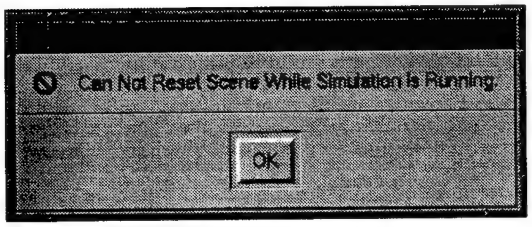

Can Not Reset Scene While Simulation Is Running

Figure 4-15. AMES Error Dialog. AMES displays different messages depending on the error encountered by the user.

4.3.2. Interface with OPEC files. A main requirement for AMES is to be compatible with

OPEC data files. The following is a description of the OPEC files that are relevant to AMES.

4.3.2.1. Trajectory and Burst Point Data Files. To setup an endgame scenario, AMES can load and use an OPEC trajectory file. AMES uses the data included in a trajectory to set the initial simulation conditions. AMES can also load an OPEC burst data file. AMES uses the information in this file to render a green sphere at the given burst point. The importance of showing this point is that it can be compared to an ENCOUNT relative miss point giving fuzing engineers a better picture of the endgame.

An OPEC trajectory file consists of the following information: The first number in the file is the number of trajectories the file contains. The file than contains the units for the target's velocity, the missile's velocity, the aim point and the miss point. These units can be either *M* for meters, *I* for inches, *C* for centimeters or *F* for feet. The file then lists the trajectory information. Each trajectory in the file consists of the information in Figure 4-16 (4:B-20):

El	Az	Vt	Vm	Ax	Ay	Az	MVx	MVy	MVz	AOA-El	AOA-Az	Ty	Tp	Tr	F1	F2	F3

Figure 4-16: OPEC Trajectory Data File Information.

For more information on each of these variables, see the OPEC User's Manual (4:B-20).

The burst point data file is set up in the same way as the trajectory file. The first number is the number of burst points in the file. The next three characters are the units for the missile and target velocities and the burst point. The units are the same as described for the trajectory file. The file then contains a list of burst point information lines. These lines coorespond to the same line in the trajectory file created by OPEC. Each line in the burst point file consists of the information in Figure 4-17.

Az	El	Roll	AOA-Az	AOA-El	Mis-Speed	Targ-Speed	BP.x	BP.y	BP.z	Ty	Tp	Tr

Figure 4-17: OPEC Burst Point Data File Information.

For more information on each of these variables, see the OPEC User's Manual (4:B-22).

4.3.2.2. Warhead Data Files. The fragment trajectories AMES creates depend on the data loaded from either an OPEC point source or explicit warhead file. AMES can load either type of warhead file and uses the information contained in the file to simulate the number, direction and speed of the warhead fragments. Currently, when the user selects the option to view the simulated fragment trajectories, AMES uses the fragment information the user most recently loaded.

4.3.2.2.1. Explicit Warhead. The OPEC explicit warhead file describes a fragment by fragment listing of the trajectory and characteristics each has. Just as with the trajectory and burst point files, the first line contains the number of fragment rays and the units of the fragment information. For each fragment in the explicit warhead file, the information in Figure 4-18 is given.

x_0	y_0	z_0	σ_{x0}	σ_{y0}	σ_{z0}	v_x	v_y	v_z	σ_{vx}	σ_{vy}	σ_{vz}	sp	σ_{sp}	m	σ_m	ρ	sf	t	d	w	l/d

Figure 4-18: OPEC Explicit Warhead Information.

For more more information on each of these variables, see the OPEC User's Manual (4:B-22).

4.3.2.2.2. Point Source Warhead. The OPEC point source warhead file describes the warhead in terms of the fragment's characteristics in a specific polar zone of the warhead. Among these characteristics are the number of fragments and the velocity of all the fragments that will emanate from the polar region. The file's first line contains the number of zones in the warhead and the parameter's units. Again, these units are the same as described in the trajectory file section. Each line in the point source warhead file then describes a polar coordinate zone located on the warhead. The data on each line consists of the information in Figure 4-19.

nl	sa	sv	ea	ev	ma	dn	sf	n2	d	w	1/d

Figure 4-19: OPEC Point Source Warhead Information.

For more information on each of these variables, see the OPEC User's Manual (4:B-16).

4.4. *Inventor Database Design.*

The Inventor scene database is a tree structure built using methods associated with Inventor objects inherited from the SoGroup primitive. These methods, addChild and insertChild, add nodes in a tree format to create a database that is applied to an Inventor viewer for rendering. To render the objects, Inventor traverses the tree from top to bottom and left to right. Therefore, objects to the right and below a node are effected by that node's attributes. The following is a discussion of AMES' Inventor tree. Reference Appendix A for diagrams of AMES' database.

4.4.1. *Main Inventor Tree.*

4.4.1.1.1. Viewing Windows and Cameras. I designed AMES' Inventor tree from top to bottom beginning with the four root nodes for each of the SoXtExaminerViewer's used in AMES. Each of these root nodes has as their first child a different SoPerspectiveCamera. These cameras define a different viewpoint for each viewer. Only one camera can be applied to a view window. This limitation is the reason the scene graph has four root nodes.

4.4.1.2. Selection Nodes and Callbacks. The second child of each root node is the main AMES tree with the SoSelection *selectroot* as its root (8:541). The SoSelection node is used as the tree's root to select a particular object in the scene when the user clicks on it. To accomplish this function, *selectroot* is attached to the SoCallback's *selectionCallback* and *deselection-Callback.* The callback's are also inserted as children to *selectroot. SelectionCallback* takes the tree path returned by the Inventor viewer to the object selected by the user and surrounds the object with a SoTransformBoxManip node. This node, as described in Chapter 3, allows the user to manipulate the surrounded object. Included in this code, is a decision making process that only lets the user select certain object types. For example, the user can not select a sphere object such as the burst point sphere. When a user clicks on the background, the *deselectionCallback* routine is called. This function removes the SoTransformBoxManip from the scene leaving the object in its new position and orientation.

4.4.2. Selectroot Tree. Along with the selection and deselection callback nodes, *selectroot* has two other children. Its first child is the Inventor SoUnits node *myUnits* (8:702). This node affects the rest of the scene graph by setting the tree's unit. For AMES, this node is set to meters because of the use of GEORGE model files which are in meters. Its second child, is the SoSeparator *sim*. The following section describes this nodes children.

4.4.3. Sim Inventor Tree. *Selectroot*'s main child is the SoSeparator *sim*. *Sim* has as it first six children, SoLight nodes (8:333). Each of these nodes places a white light source at a positive and negative point along each world coordinate. *Sim*'s seventh and eighth children are *targetroot* and *missileroot*. These nodes are the separators used to transform the missile and target according to their specific endgame attributes. Finally, sim's ninth child is the *burst-PointGroup* node that is added when the user selects OPEC trajectory and burst point data to simulate.

4.4.4. Missile Inventor Tree.

4.4.4.1. Missile Transformations Nodes. The SoSeparator *missileroot* separates the nodes that manipulate the missile from the rest of the scene graph. The missile has four nodes that transform the missile according to different simulation parameters. First, the SoTransform *missileSlideTransform* is connected to the SoComposeVec3f *missileSlideDistance* (8:159). This node in turn, is connected to the SoElapsedTime engines *missileCounterX*, *missileCounterY* and *missileCounterZ* (8:227). These engines are used to simulate the missile movement in the X, Y and Z directions. They supply the *missileSlideTransform* node information to translate the missile in three dimensional space. Second, the SoTransform *missileXform* transforms the missile's initial position to the relative miss point. It is also the node that is replaced by the SoTransformBoxManip when the user selects the missile. Third, the SoMatrixTransform *missileAzElMatrixTransformation* rotates the missile according to the given azimuth and elevation parameters (8:354). Finally, the SoMatrixTransform m*issileAOAAzElMatrix-Transformation* rotates the missile according to the given angle of attack azimuth and elevation parameters. AMES does not take into account missile roll because there was no requirement to add this parameter to the simulation. These four nodes are the first, second, third and seventh children of *missileroot,* respectively.

4.4.4.2. Missile Velocity Vector Nodes. Missileroot's fourth child is the SoGroup *missilePointDraggerGroup* (8:273). This node is a place holder for the *missilePointDragger-Root* node. *MissilePointDraggerRoot* is the parent node for the missile's velocity vector parts, SoDragPointDragger *pointDragger* and an instantiated *lineGroup.* The pointDragger, as described in Chapter 3, manipulates the end of its attached velocity vector. This node's placement in the scene graph is important because it needs to be transformed by all missile transformation nodes except for the angle of attack azimuth and elevation node. The lineGroup structure draws an Inventor line and colors it according to the given SoMaterial. This line represents the velocity vector. See Appendix A, page A-4 for a diagram of the *lineGroup* structure.

4.4.4.3. Missile Fragmentation Group. Missileroot's fifth child is the SoGroup *fragmentMainGroup.* This node is a place holder for the SoSeparator *fragmentGroup.* This node is the parent of each lineGroup that represents a single fragment trajectory. It is this node, *fragmentGroup,* that is added and removed when the fragment trajectories are toggled.

4.4.4.4. Missile Coordinate System Nodes. Missileroot's sixth child is the SoGroup *missileCoordinateAxisGroup.* This node is a place holder for the node *missileCoordinate-AxisRoot.* The SoSeparator *missileCoordinateAxisRoot* has as children, three *lineGroup's.* These groups draw the three coordinate axis for the missile. Coordinate axis' visibility is toggled by adding and removing the *missileCoordinateAxisRoot* node from *missileCoordinateAxisGroup.*

4.4.4.5. Missile Wrapper Node. Missileroot's eighth child is the SoWrapperKit *mWrapper* (8:706). This node combines the tree added to its contents and treats it as one object. Currently, the FASTGEN missile file translation into Inventor produces an object consisting of multiple parts. Therefore, the SoWrapperKit is required to treat the multiple objects defined in the missile as a single object for selection purposes. AMES does not currently use the SoWrapperKit's transform part.

4.4.4.6. Missile Fuze Cone Tree. To render the multiple fixed cone fuzes, I add as a child to *mWrapper's* contents, the SoSeparator *fuzeConeRoot.* This node has as it's children either two, four or six SoSeparators depending on the number of fuze cones loaded from the AMES fuze data file. Each SoSeparator defines an inner or outer fixed fuze cone. Thus, for each fuze defined, there are two SoSeparators added to *fuzeConeRoot.*

Each SoSeparator has as children, a SoMaterial, a SoTransform and a SoCone. The SoMaterial node defines the fuze cone's color and transparency. The SoTransform node transforms the cone's tip to the fuze origin point. Finally, the SoCone defines the fuze's cone shape, depending on the range and half angle defined in the fuze input file (8:167). After calculating the transform and the cone's height and radius atrributes, the SoSeparator is added as a child to *fuzeConeRoot*.

FuzeConeRoot has one other child, the SoTransform *fuzeXform*. The SoTransform rotates all subsequent cones ninety degrees around the Y axis. This places them such that they are emanating from the missile, pointing down the missile coordinates positive X axis. Finally, it is important to note that when the user selects the option to toggle on the fuze cones, *fuzeConeRoot* is placed within the SoWrapperKit's contents. Therefore, the fuze cones act as part of the missile for selection and movement purposes.

4.4.4.7. Missile Movement Nodes. To simulate missile movement, I am using a pre-defined inventor primitive called an *engine*. The missile's SoComposeVec3f *missileSlideDistance* is connected to three Inventor SoElapsedTime engines *missileCounterX*, *missileCounterY* and *missileCounterZ*. These three engines attach to the X, Y and Z portions of the SoComposeVec3f node. When the user selects play on the control window, the SoElapsedTime engine's are activated and output a SoSFTime result to each of the SoComposeVec3f's portions. This output causes the X, Y and Z components to change thus updating the translation factor of *missileSlideTransform*, resulting in the missile moving through the scene.

4.4.5. Target Inventor Tree.

4.4.5.1. Target Transformation Nodes. The SoSeparator *targetroot* separates the nodes that transform the target from other scene graph parts. It is designed in the same manner as *missileroot*. Like *missileroot*, *targetroot* has multiple transformation nodes responsible for changing different aspects of the target for a simulation. First, *targetroot* has the SoTransform *targetSlideTransform* attached to the SoComposeVec3f *targetSlideDistance*. This node, in turn, is attached to the three SoElapsedTime engines *targetCounterX*, *targetCounterY*, and *targetCounterZ.* As with the missile, these engines are responsible for moving the target in each of the three dimensions. Next, *targetroot* has the SoTransform *targetXform*. This node is a place holder for the SoTransformBoxManip. It is replaced when the user selects the target with the mouse. Third, the SoMatrixTransform *targetYawPitchRollMatrixTransformation* rotates the target model according to the yaw, pitch and roll given in the endgame parameters. These three nodes are the first, second and fifth children of *targetroot,* respectively.

4.4.5.2. Target Coordinate System Nodes. *Targetroot*'s third child is the SoGroup *target-CoordinateAxisGroup*. This group is a place holder for the addition and deletion of the *lineGroups* used to draw the Inventor lines defining the target's coordinate axis. These three groups are children of *targetCoordinateAxisRoot*. It is this node that is added and deleted from the tree when the user toggles the coordinate axis.

4.4.5.3. Target Velocity Vector Nodes. As with the missile, the target has a SoGroup node, *targetPointDraggerGroup*, holding the place for the target's velocity vector parts. This node is *targetroot's* fourth child. The velocity vector's parts, the SoDragPointDragger *pointDragger* and the *lineGroup*, are added as children to *targetPointDraggerRoot*. When the user toggles the target's velocity vector, the *targetPointDraggerRoot* is added or removed as a child of *targetPoint-DraggerGroup*.

4.4.5.4. Target SoWrapperKit Node. *Targetroot*'s fifth child is the SoWrapperKit *target-Wrapper*. This SoWrapperKit node, like the missile, is used to combine the target model parts so they act as one object for selection purposes. Again, the SoWrapperKit's transform part is not used.

4.4.5.5. Target Movement Nodes. As with the missile, the target is connected to three SoElapsedTime nodes. When these nodes are turned on, they change the X, Y and Z translations of the *targetSlideTranslation* node causing the target to move through the scene.

4.4.6. BurstPointGroup. This group consists of three nodes used to depict the OPEC burst point as a sphere in three dimensional space. *BurstPointGroup's* first child is *burstPointTransform.* This node translates the sphere to the burst point's position in target coordinate space. BurstPointGroup's second child is *burstPointMaterial.* This node colors the sphere green. The final child is *burstPointSphere.* This node is a pre-defined Inventor SoSphere node. The *burstPointGroup* is only added to the scene when OPEC data is selected. ENCOUNT data has no burst point defined.

4.4.7. SoNodeSensors. To calculate position information to output to the target and missile position window, I have used an Inventor node called a SoNodeSensor (8:427). This sensor detects when the node to which it is attached changes. When it detects a change, the sensor invokes a callback routine. The following is a list of sensors I used, their connections, and the functions they call.

1. *myMissileSlideTranslationChangedSensor* - Connects to missileSlideTranslation and calls getMissilePos to calculate the missile's new position in world coordinates when the SoElapsedEngines are turned on.
2. *myMissileXformChangedSensor* - Connects to missileXform and calls getMissilePos to calculate the missile's new position in world coordinates when it is translated to the relative miss point.
3. *mytbManip_missileChangedSensor* - Connects to tbManip_missile, the SoTransform-BoxManip that replaces missileXform in the Inventor tree. It calls getMissilePos to calculate the missile's new position in world coordinates when the user moves the missile using the manipulator.
4. *myTargetSlideTranslationChangedSensor* - Connects to targetSlideTranslation and calls getTargetPos to calculate the target's new position in world coordinates when the SoElapsedEngines are turned on.
5. *myTargetXformChangedSensor* - Connects to targetXform and calls getTargetPos to calculate the target's new position in world coordinates when it is translated.

6. *mytbManip_targetChangedSensor* - Connects to tbManip_target, the SoTransform-BoxManip that replaces targetXform in the Inventor tree. It calls getTargetPos to calculate the target's new position in world coordinates when the user moves the target using the manipulator.
7. *myTargetSlideTranslationChangedSensor2* - Connects to targetSlideTranslation and calls getVulnerablePos to calculate the vulnerable point camera's new position in world coordinates when the SoElapsedEngines are turned on.
8. *myDraggerSlideTranslationChangedSensor* - There are two instances of this sensor. One is connected to the missile's *pointDragger* and the other connected to the target's *pointDragger*. It calls the velocity vector method callTrueGetVelocity to calculate the velocity vector's new magnitude when the user manipulates the *pointDragger*.

4.5. Endgame Simulation Modeling.

To understand the reasons behind my decisions, it is important to understand the endgame data sets and how they interact with each other. This section provides background information on general endgame parameters and fixed cone fuzing used in AMES.

4.5.1. General Endgame Parameters. Many parameters are necessary to simulate a missile endgame. This section covers what those parameters are, and important notes on how they are used to manipulate the endgame entities.

First, the target is rotated according to a given yaw, pitch and roll. The missile, in turn, is rotated by a given azimuth and elevation. These parameters are combined to create a rotation transformation for each entity from the model coordinate systems to the world coordinate system. It is important to note that the target's rotation matrix is created by post-multiplying the yaw, pitch and roll matrices (in this order), to an identity matrix. Equation (2) shows the correct order of matrix multiplication.

$$Rotation = I * Y * P * R \tag{2}$$

For the missile, the azimuth and elevation (in this order), are post-multiplied to an identity matrix to create the missile's rotation matrix. As stated before, the missile is not given a roll parameter while the azimuth corresponds to yaw and elevation corresponds to pitch. The missile's roll is assumed zero. Equation (3) shows the correct order of matrix multiplication.

$$Rotation = I * A * E \tag{3}$$

Both the missile and target are given a velocity vector magnitude and direction. The target's velocity vector is assumed to extend down the target's coordinate system's positive X axis with the given magnitude. The missile's velocity vector is rotated according to the missile's azimuth and elevation with the given magnitude.

There are also multiple points defined in an endgame simulation. One point is the target aim point. This point defines the ideal impact point on the target for the given azimuth and elevation and corresponds to a zero miss distance. The aim point is always given with respect to the target coordinate system. Another point is the miss point. This point is defined as a distance along the miss plane where the missile actually passes through given its azimuth and elevation parameters. This miss point is defined in the relative trajectory system. See Appendix C for a discussion of the relative trajectory system. The trajectory defined by the miss point with the given azimuth and elevation is parallel to the trajectory defined by the aim point with the given azimuth and elevation.

WL/MNMF engineers also define a missile tracking point that is analogous to the target aim point. This point is defined with respect to the missile coordinate system and is the point that engineers use to follow along the relative miss trajectory. They also define a missile arbitrary point. This point is used to see how close the missile comes to hitting the target. When the missile's tracking point is drawn at the relative miss point, the arbitrary point corresponds to the missile's burst point. In other words, this arbitrary point is usually defined to be the warhead's forward end.

Finally, WL/MNMF engineers define a rotation point for both the target and missile. This point is used to translate the target and missile coordinates in object space such that the rotation matrices will rotate the objects accordingly. This rotation point was created because most FASTGEN models are not defined using the object space origin as the model's center. Therefore, standard rotation matrices rotate the object around this origin point resulting in an incorrect target yaw, pitch and roll or missile azimuth and elevation.

4.5.2. Fixed Cone Fuzing. A fixed cone fuze is defined by an interior half angle, an exterior half angle and the range the fuze sensor extends. Figure 4-20 shows these variable configurations. To create the fuze cones, I used the Inventor SoCone primitive. This primitive has as attributes, a radius and height. To calculate the correct radius and height for both the interior and exterior fuze cone given the angle and range (e.g., hypotenuse), I used basic trigonometry. Equations (4) and (5) are calculations for the radius and height and a cone. These calculations are done for both the inner and outer cones.

$$\text{radius} = \sin(Fi_x) * R \tag{4}$$

$$\text{height} = \cos(Fi_x) * R \tag{5}$$

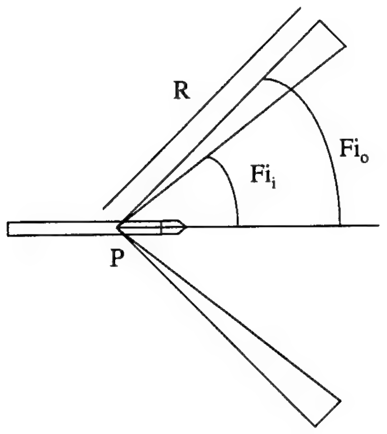


Figure 4-20: Description of a fuze cone. R is the fuzes range, Fi_i is the cone's inner angle, and Fi_o is the cone's outer angle

To place the cones in the correct position, I needed to calculate a rotation and transformation to put each cone's tip onto the fuze's origin as given in the AMES fuze data file. To accomplish this function, I first rotate each cone 90 degrees around the missile's Y coordinate axis. This rotation brings the cone into alignment with the missile's direction. The cones are then translated according to the difference between the fuze origin point and the cone's tip. I calculate the cone's tip point by first calculating the cone's center and subtracting one-half of the cone's height from it in the X coordinate.

4.5.3. AMES Fixed Fuze Cone Input File. To input the fixed fuze cone information, the user must create an AMES fuze data file as show in Figure 4-21. The first number in this file is the number of fixed fuze cones to be displayed. This number can be between one and three. The rest of the first line is ignored and can be used for comments. The second line is the fuze origin X, Y and Z coordinate point. This point is given in missile coordinates. Finally, for each fuze cone, the user defines the inner angle, the outer angle and the fuze cone's range. Again, these parameters are shown in Figure 4-20.

```
3 // x, y, z of missile point, inner angle, outer angle, range
0 0 0
30 33 30
65 67 30
82 85 30
```

Figure 4-21: An example of the fixed cone fuze data file.

4.6. Conclusion.

AMES's implementation has brought together the graphics power of the Inventor library with the windows usability of Viewkit and Motif to create a program for missile fuzing engineers to view the results of their simulations. Although the ground work has been laid, there is much more that can be added to further AMES's capabilities. Suggestions for future work are covered in Chapters 5 and 6.

5. Results

5.1. Introduction.

To measure the results of AMES, I will compare the initial requirement set with the requirements met during development. To this end, I break this chapter into two sections. Section 5.2 lists satisfied requirements; Section 5.3 lists requirements that were not satisfied. Included in my discussion, I cover how a change in requirements mid-way through the thesis cycle modified my focus, making some initial requirements less important and more appropriate for future work. I end this chapter with a discussion of how the current AMES implementation can add to the missile endgame simulation process to support improvements in missile fuzing and warhead technology.

5.2. Initial Requirements.

In Chapter 3, I described the initial set of AMES requirements. Initially, the main purpose was to build a three dimensional graphics program that a user could use to create a missile endgame scenario using a mouse driven user interface. Given this capability, the program's purpose was to

1. explain air-to-air fuzing concepts to personnel,
2. experiment with sensor coverage concepts, and
3. graphically demonstrate the results of fuze simulations.

The initial crux of this project was to give the user as much direct manipulative capability over the simulation parameters as possible. Included in the requirement was the capability for the user to select a missile and target model, place the models into a three dimensional space and manipulate the endgame parameters using either keyboard or mouse input. Required modifiable parameters initially included the missile and target velocity vectors, the missile's and target's position, the target's yaw, pitch and roll, the missile's azimuth and elevation and the missile's angle of attack azimuth and elevation. Mid-way through the thesis cycle though, the capability to manipulate the endgame parameters by hand was rejected in favor of changing them according to pre-defined OPEC and ENCOUNT scenario data files. Because of this change, I only implemented the capabilities to manipulate the translation and rotation of either the missile and target and velocity vector manipulation for each model.

Other initial requirements were to simulate missile and target movement, simulate fixed and skewed cone fuze coverage patterns, and simulate warhead fragment coverage patterns. New requirements included the capability to load either OPEC or ENCOUNT trajectory data. These requirements were implemented. The full set of requirements I completed during AMES design and implementation are shown in Table 5-1.

5.2.1. User Interface. AMES graphical interface provides all necessary functionality for the user to manipulate and view an endgame scenario. As described in Chapter 4, it provides a menu to import data and change various attributes of the endgame scenario. AMES also provides multiple capabilities to manipulate the user's viewpoint as well as the draw style for the models in the scene. The user's interface is easily learnable, efficient in its interaction with the endgame entities, and memorable since it uses a standard menu interface that refers to well known endgame information.

5.2.2. Simulation Cameras. AMES provides the user with multiple camera positions to view the simulation. These positions include the main window camera, a camera tethered to the missile, a camera thethered to the target, a vulnerable point camera defined by the user and finally three position view windows the user can activate along with the main window. An

Fulfilled Requirements	Requirement Description	Type
User Interface	Provide capability to insert and manipulate missile and target model.	Init
	Provide capability to manipulate a velocity vector for target and missile.	Init
	Provide windows for feedback (position, velocity).	Init
Simulation Cameras	Provide capability to manipulate camera position to view simulation from different views.	Init
	Provide capability to define a vulnerable camera point in target coordinate space and give capability for user to rotate camera view point.	Mid
	Provide cameras that tether to missile and target and follow object through simulation.	Mid
	Provide capability to center camera on scene when objects move beyond cameras view.	Init
	Add multiple windows so user can view simulation from different points simultaneously.	Init
	Provide capability to change the viewing frustum's height angle.	Mid
Data Input	Provide capability for file input and keyboard input of endgame parameters.	Init
	Provide capability to load ENCOUNT and OPEC style trajectory information.	Mid
	Provide capability to load OPEC burst point, explicit warhead, point source warhead and OPEC output files.	Mid
	Provide capability to load FASTGEN converted target and missile models.	Init
	Provide capability to load in AMES style fuze sensor data.	Init
	Provide capability for user to input endgame parameters and object position and velocity information with the keyboard.	Mid
Scaled time simulation	Provide capability to play, rewind, fast forward, pause and stop an endgame simulation.	Init
Endgame Setup	Provide capability to select from OPEC trajectory information in loaded file.	Mid
	Provide capability for user to change ENCOUNT endgame parameters.	Mid
Fuze Sensors	Show fuze coverage pattern for one to three fuze sensors.	Mid
	Provide capability to define the fuze sensor origin in missile coordinates.	Init
Fragmentation Patterns	Provide capability to show point source or explicit warhead data.	Init
	Color code data according to point source polar region.	Init
	Calculate trajectories according to current missile velocity.	Init
Coordinate Axis	Provide current missile and target coordinate axis.	Init
Relative Trajectories	Provide capability to show relative trajectories.	Mid
Target Damage	Provide color coded target damage feedback according to OPEC output data.	Mid
Edit Background	Provide capability to edit background color of four view windows.	Init
Reset Scene	Provide capability to reset the endgame scene. Remove all rotations and translations placing target and missile at origin.	Init
Toggle Options	Provide capability to toggle the view options including missile and target velocity vectors, fuze sensor coverage, relative trajectories, warhead fragmentation, target damage and three position view windows.	Init
FASTGEN conversion	Provide program to convert FASTGEN models into Inventor models.	Init
Feedback dialogs	Provide feedback dialogs for user actions.	Init
Busy state feedback	Provide a busy state for when program is processing calculations and no user interaction is allowed.	Init

Table 5-1: AMES requirements listing.
Init = Initial requirement; Mid = Midterm Requirement change.

important capability for the user to change the viewpoint are six thumbwheels that manipulate the position and focal point of the main window camera. The multiple cameras and the capability to change their positions and focal points, allows the user to view the simulation from virtually any point.

5.2.3. Data Input. AMES was required to be compatible with OPEC and ENCOUNT data files to create an endgame scenario. The requirement of loading each of these files was fulfilled by using standard C++ file I/O streams. Each data set is stored in an class object instantiation used by AMES.

Along with loading information from data files, AMES was also required to change scenario parameters with keyboard input. I fulfilled this requirement with the ENCOUNT data input window. The user can also change each endgame parameter individually and update the scenario without rerunning the program.

A final requirement was to input any Inventor missile or target model into the simulation. This requirement was fulfilled by using the Inventor SoInput node. This node loads an Inventor formatted data file such that the polygons or objects in the file can be directly added to an Inventor scene database.

5.2.4. Scaled time simulation. To provide scaled time simulation, I created a control window and added multiple SoElapsedTime engines to simulate target and missile motion. The motion can be controlled to provide play, fast forward, rewind, pause and stop capability.

5.2.5. Endgame Setup. AMES includes the capability to view the scenario of any trajectory in an OPEC trajectory data file. This requirement stems from OPEC's lack of this functionality. The user can also view the currently loaded OPEC data with the OPEC list window. Along with OPEC data, the user can also use the ENCOUNT data window to change the endgame scenario. The user can take advantage of this functionality to toggle between ENCOUNT and OPEC endgame scenarios.

5.2.6. Fixed Fuze Sensor Coverage. To simulate multiple fixed fuze sensor coverage patterns, AMES uses Inventor SoCone nodes. These nodes are manipulated according to the interior and exterior half angles and range data contained in the AMES fixed fuze sensor data file. This capability allows engineers to visualize how fixed fuze cone patterns interact with a target.

5.2.7. Warhead Fragmentation Patterns. To simulate warhead fragmentation patterns, I used Inventor SoLineSets to define each fragment's trajectory. When viewed, they display the total coverage pattern for the warhead. It is important to note that this requirement did not include scaled time fragment trajectory simulation. In fact, WL/MNMF specifically requested that the fragmentation's path not be simulated. Their main emphasis is not to see the individual fragmentation trajectories in motion but to view the coverage pattern they create.

5.2.8. Model Coordinate Axis. Each model has a coordinate axis defined by three Inventor SoLineSets. These lines are manipulated according to the models endgame parameters such that they portray the current model coordinate system. This requirement was prompted by the desire to convey the same information as ENCOUNT.

5.2.9. Relative Trajectories. To view an endgame's optimal and actual relative trajectories, I defined two Inventor SoLineSets. The slopes for each line are calculated using the ENCOUNT code provided by WL/MNMF. The lines are extended from their respective points, the aim point and the miss point, by one hundred meters in both directions. Again, this requirement was prompted by the desire to convey information displayed by ENCOUNT.

5.2.10. Target Damage Feedback. Target damage feedback is supplied by color coding the damaged components according to a graduated color scale. When the user selects the *show* *target damage* option, the target's damaged parts are colored accordingly. This feedback provides an excellent method to visualize the outcome of an OPEC endgame simulation.

5.2.11. Edit Background Color. The capability to change the background color was added to enhance AMES usability. It provides one more variable for the user to change to accomplish their objective of demonstrating endgame and fuzing methods to personnel.

5.2.12. Reset Scene. AMES has an additional feature that resets the endgame scenario. This feature stems from my experience in creating a simulation scenario. After manipulating the scene and running a simulation, the user may have a need to reset the scene and start again.

5.2.13. Toggle Options. The user has control over which components are shown during a simulation. Depending on the simulations requirements, the user can toggle multiple options as described in Chapter 4. This control supports AMES dual purpose as described in Chapter 6.

5.2.14. FASTGEN to Inventor File Conversion. As described in Chapter 3, I initially fulfilled the requirement of using FASTGEN models in AMES by writing a program to convert OPEC GEORGE files to Inventor. However, during my thesis effort, Capt. Jeff Bush implemented an improved version of the conversion process. His program directly converts a GEORGE file into an Inventor file. This improved conversion process is capable of producing multiple level of detail models for use in AMES (11:1-2).

5.2.15. Feedback Dialogs and Busy States. Finally, I added multiple error and information dialog boxes to provide improved user feedback as to AMES' current state. Along with the dialog boxes, I also added busy states for when AMES is making calculations. These busy states restrict the user's interaction with AMES to ensure the correct calculations are made for the endgame scenario.

5.3. *Unsatisfied Requirements.*

The requirements change mid-way through my thesis adjusted my focus from the initial requirements to a new set of requirements. Thus, I decided some of the initial requirements were better left for future work. These requirements are listed in Table 5-2.

Unsatisfied Requirements	Requirement Description
Skewed Cone Fuze Coverage	Incorporate MSFUZE algorithm to simulate skewed cone fuze coverage
Coordinate Feedback	Provide capability to have position feedback in either target centered, missile centered or world coordinate centered
	Provide capability to switch the units for feedback and data input (Meters, Inches, Feet, Centimeters)
Velocity Vector Orientation	Provide feedback as to velocity vectors orientation to missile and target
Time Passed Feedback	Provide feedback on time passed during simulation

Table 5-2: Unsatisfied AMES requirements

5.3.1. Skewed Cone Fuze Coverage. One requirement not implemented was the incorporation of the MSFUZE algorithm supplied by WL/MNMF. This algorithm defines the skewed cone fuze sensor pattern. The algorithm is an interface between the MFOTS algorithm and SHAZAM. It transforms data from SHAZAM along with an error measurement to create measured values for the fuze algorithm with the MSFUZE program (20:1). The integration of this FORTRAN program into AMES should be a top priority for future work.

5.3.2. Coordinate Feedback. Another requirement left for future work, is the capability to switch amongst a target centered, missile centered or world coordinate centered feedback option. The information displayed in the position windows would be relative to the chosen center point.

5.3.3. Velocity Vector Orientation. To provide the user with more feedback as to an object's velocity vector's relative position, the orientation data should be displayed. This information gives the user a better feel as to the direction the missile and target move when the simulation is run.

5.3.4. Model Manipulation. Currently, there is no capability to select and manipulate certain endgame simulation parameters using the mouse. The user can change the missile's azimuth and elevation using the Inventor manipulator. However, the user can not select the missile itself and rotate the angle of attack elevation and azimuth. The same problem exists with the target. There is currently no capability to change just the target's yaw, pitch and roll without rotating its coordinate system. By adding these capabilities, the user can create endgame simulation scenarios without using OPEC or ENCOUNT generated data.

5.3.5. Feedback on Simulation Time. Another requirement was to provide a feedback window on how much time has past during the simulation depending on a selectable speed. This feedback would show the number of milliseconds that past during scaled time simulation.

5.4. AMES Use for Endgame Simulation Modeling.

AMES initial purpose was to give a missile fuzing engineer the capability to visually create a missile endgame scenario. This initial purpose grew to encompass the capability to view fragmentation trajectories, target damage and missile fuze cone interaction with the target. The addition of these new requirements though, gives the user more flexibility to view multiple parts of an endgame scenario. This flexibility in simulating multiple endgame scenario phases makes AMES an excellent tool for fuzing, warhead and aviation engineers to view endgame scenario data. With AMES strengths in mind, this section describes the endgame simulation process and how AMES can fit into this process.

5.4.1. Endgame Simulation Process. The missile endgame simulation process as practiced by WL/MNMF begins with an idea. This idea, whether it is a new fragmentation pattern or a new method for fuze sensor coverage, needs to be formulated and tested for viability. Due to the expense of building a prototype and conducting a live test for each new proposal, the idea first needs to be tested using other evaluation systems such as the OPEC simulation program. Multiple OPEC simulation runs can produce a confidence interval as to the new idea's effectiveness in improving a missile's capability. Using AMES, engineers can evaluate visually, the probable interactions of the proposed system with a target, given the data produced by OPEC.

AMES is designed as a baseline for visual simulation of newly formed ideas. It is flexible enough to show fixed fuze cone patterns as well as new point source warhead definitions. It is also an excellent platform to incorporate previously designed software programs such as MSFUZE to visually simulate skewed cone fuze coverage patterns. Therefore, the lack of scaled time, graphical motion feedback of previous programs for new simulation ideas is filled by AMES.

5.5. Conclusion.

AMES is a success in that it provides a platform to view and simulate different combinations of fixed fuze sensor options and warhead fragmentation patterns. The program expands the set of tools the fuze and warhead engineer can use to improve his trade. However, AMES is far from complete. As a baseline for graphical feedback, AMES can be expanded to incorporate other endgame simulation code and simulate its results. Along this vein, I discuss future work recommendations for AMES in Chapter 6.

6. *Conclusions and Recommendations.*

6.1. *Introduction.*

My work on AMES produced a baseline program capable of simulating fixed fuze cone sensor coverage, warhead fragmentation and target damage of an endgame scenario. It also simulates in a scaled time environment, the interaction of these attributes. The work I have completed fulfills the requirements as listed in Chapter 5. With these requirements accomplished, this chapter describes future work for improving AMES functionality.

I begin this chapter in Section 6.2, by describing how AMES contributes to the endgame simulation process. This discussion describes how AMES improves the graphical output of both ENCOUNT and OPEC. In Section 6.3, I cover recommendations for future work. I finish the chapter with a wrap-up of my thesis work and some final comments.

6.2. *AMES Contributions.*

As described in Chapter 5, AMES is designed to improve endgame graphical feedback by creating a three dimensional view of an endgame scenario. Instead of looking at pages of output data, an engineer can use AMES to view simulation output. Although WL/MNMF already has two tools to visually display certain endgame data, AMES has many advantages over both OPEC and ENCOUNT graphical output. These advantages include

1. the capability to view fixed fuze cone sensor coverage patterns,
2. the capability to view warhead fragmentation coverage patterns,
3. the capability to run a simulation to view the interaction of the missile's kill mechanism's with the target,
4. the capability to create an endgame simulation using the mouse,

5. the capability to view the simulation from multiple points,
6. the flexibility to incorporate other endgame simulation code, and
7. compatibility with both OPEC and ENCOUNT data files.

This section describes each advantage AMES has over OPEC and ENCOUNT. In this context, I describe how these advantages enhance the capabilities of an engineer to analyze an example missile endgame problem.

6.2.1. Improved Endgame Simulation Process Over OPEC. OPEC is designed to complete the calculations needed to produce a probability of kill for an endgame encounter. Its main focus was not to graphically display the results. This lack of focus is seen in the Intel platform choice for which OPEC is designed. This computer platform has little graphics hardware support to create vibrant feedback of the endgame scenario. Instead, OPEC's visual feedback has very low resolution and provides the user with little detail as to the simulation's results. AMES fills the void left by OPEC's lack of high resolution graphics by showing an endgame scenario with both fuze sensor coverage patterns and warhead fragmentation patterns.

AMES also gives engineers the capability to experiment with different warhead fragmentation patterns. Using AMES, the engineer can not only view the coverage pattern of any OPEC point source or explicit warhead data set but also can change the endgame's parameters to see how the fragments disperse given these changes. For example, when the engineer stretched or changes the missile's velocity vector, the new fragmentation pattern is calculated to accommodate the vector's magnitude.

6.2.2. Improved Endgame Simulation Process Over ENCOUNT. The ENCOUNT code written by WL/MNMF was incorporated into AMES to provide the same visual feedback created by this program. Included in this feedback are the endgame's actual and optimal relative trajectories and the target and missile coordinate systems. AMES adds to the visual feedback supplied by ENCOUNT with the additions described in Section 6.2. AMES also improves on *Ivaview*, the program used to view ENCOUNT output, in its capabilities to view and endgame scenario. AMES not only provides manipulative capability to view the simulation from virtually any point but also capability to change any of the endgame scenario's parameters.

6.3. AMES Recommendations and Future Work.

Although AMES is a solid start for viewing and simulating an endgame scenario, there is still work to be done to improve the program. This section describes future work recommendations for AMES.

6.3.1. Unsatisfied Requirements. In Chapter 5, I listed five requirements that were not completed. These five requirements should be the initial focus of future work done on AMES. These requirements are

1. the incorporation of the MSFUZE skewed fuze cone algorithm,
2. providing multiple coordinate center feedback options,
3. providing velocity vector orientation feedback relative to the model,
4. providing speed control over simulation, and
5. providing feedback on time past during a simulation.

These five points complete the initial requirements to fully simulate all aspects of an endgame scenario.

6.3.2. Modifiable Level of Detail. AMES has a dual purpose. First, it is to be used by missile fuze and warhead engineers to test new ideas and to view missile endgame simulation data. Second, it is designed to graphically demonstrate fuzing techniques to personnel unfamiliar with these concepts. To this end, there should be different level's of detail for missile and target models for each of these purposes. For example, engineers who need to see the damage done to a target should use a fully converted target model with all the parts. However, to view a simulation of fuze cone or warhead interaction, a lower level of detail model could be used since the interior detail is unnecessary. The less detailed model will improve the scaled time simulation while still showing how the missile's kill mechanisms interacts with the target.

6.3.3. Add Video Recording Capability. Another requirement for AMES is the capability to video tape a simulation. The process of video taping the simulation should be easy to use and compatible with the systems in use at WL/MNMF.

6.3.4. Recording a Simulation on Disk. In addition to a video recording, AMES must save a simulation to disk. With this capability, the user can later load the file and recreate the same simulation.

6.3.5. Capability to Change Simulation Speed. Currently, AMES runs its simulations at a default speed given to it by the velocity vectors of each model. An improved method of simulation is to allow the user to increase or decrease the speed of the simulation with respect to time.

6.3.6. Rendering Shock Wave Created by Warhead Detonation. A final addition to AMES would be the capability to view the shock wave created by a warhead detonation. This shock wave affects the total damage incurred by a target and would increase the simulation's realism.

6.3.7. Consolidation of Endgame Simulation Code with AMES. A suggestion for future work is to incorporate the OPEC endgame simulation software into AMES. This addition would eliminate the need to run OPEC on a PC and then transfer the data files to a Silicon Graphics work station for use in AMES. The user could model a single encounter using OPEC code without appealing to an external program. The data could then be directly loaded into the AMES view window.

If this incorporation is not feasible, then the next step would be to port AMES from the Silicon Graphics platform to the PC platform. Currently, Inventor libraries exist to integrate with Windows 95. Before such a port is completed however, an extensive study on the AMES response time on an Intel platform should be performed.

6.4. Conclusion.

The AMES project was proposed to give the fuzing engineer a new tool in creating and visually simulating new missile endgame encounter parameters. In these simulations, the engineer could test new fuzing techniques as well as warhead fragmentation patterns to get a better feel for their effectiveness. AMES fulfills these requirements. It improves the endgame simulation process by giving engineers a tool to see the results of endgame scenarios. Thus, with AMES, engineers are better equipped to invent and produce the next generation of weapon systems for the Air Force to fly, fight, and win!

A.1 . Inventor Scene Database Design.

Figure A-1 shows AMES' top Inventor tree. The three cameras at the bottom of the figure define the attachable cameras to the missile, the target and the target's vulnerable point. These cameras are placed into the tree when the user selects to attach to that camera. Reference Chapter 4 for a complete description of this tree.

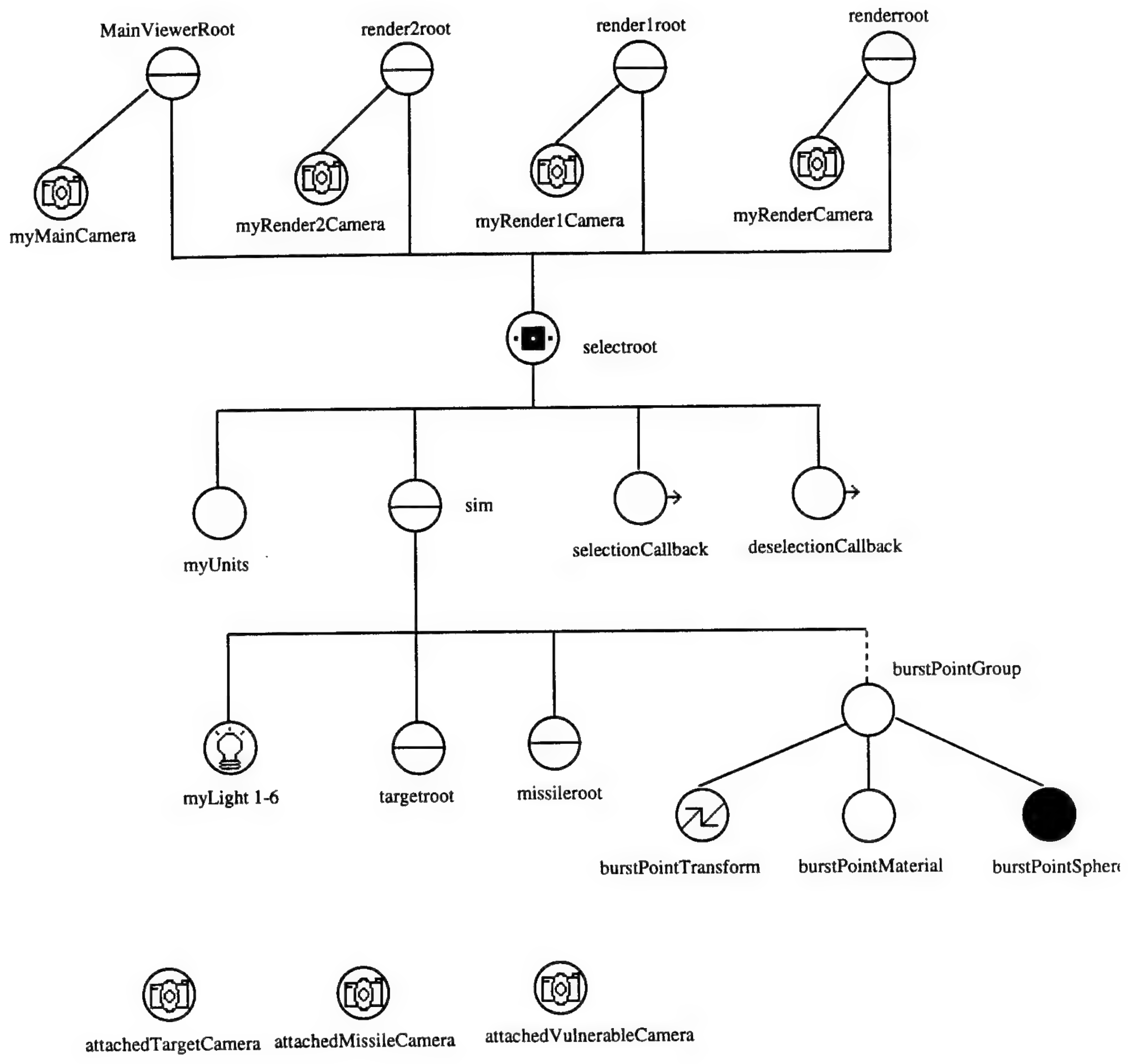


Figure A-1: AMES Inventor Database Tree

A.2. Missile Scene Database Subgraph.

Figure A-2 depicts the missiles Inventor subtree. missileroot is a child of sim as shown in Figure A-1. Reference Chapter 4 for a complete description of this tree.

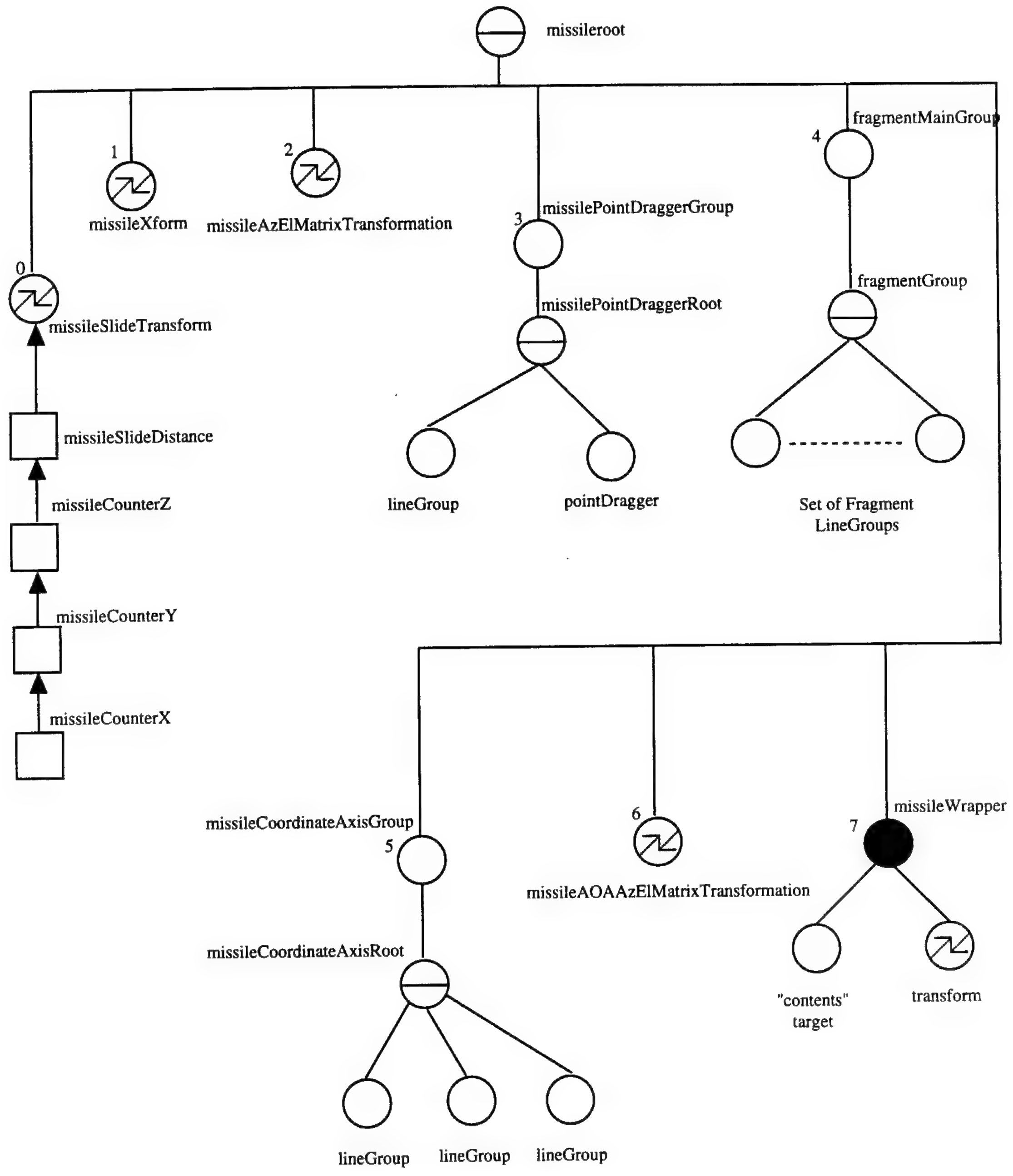


Figure A-2: AMES Missile Database

A.3 . Target Scene Database Subgraph.

Figure A-3 depicts the target's Inventor subtree. Reference Chapter 4 for a complete description of this tree.

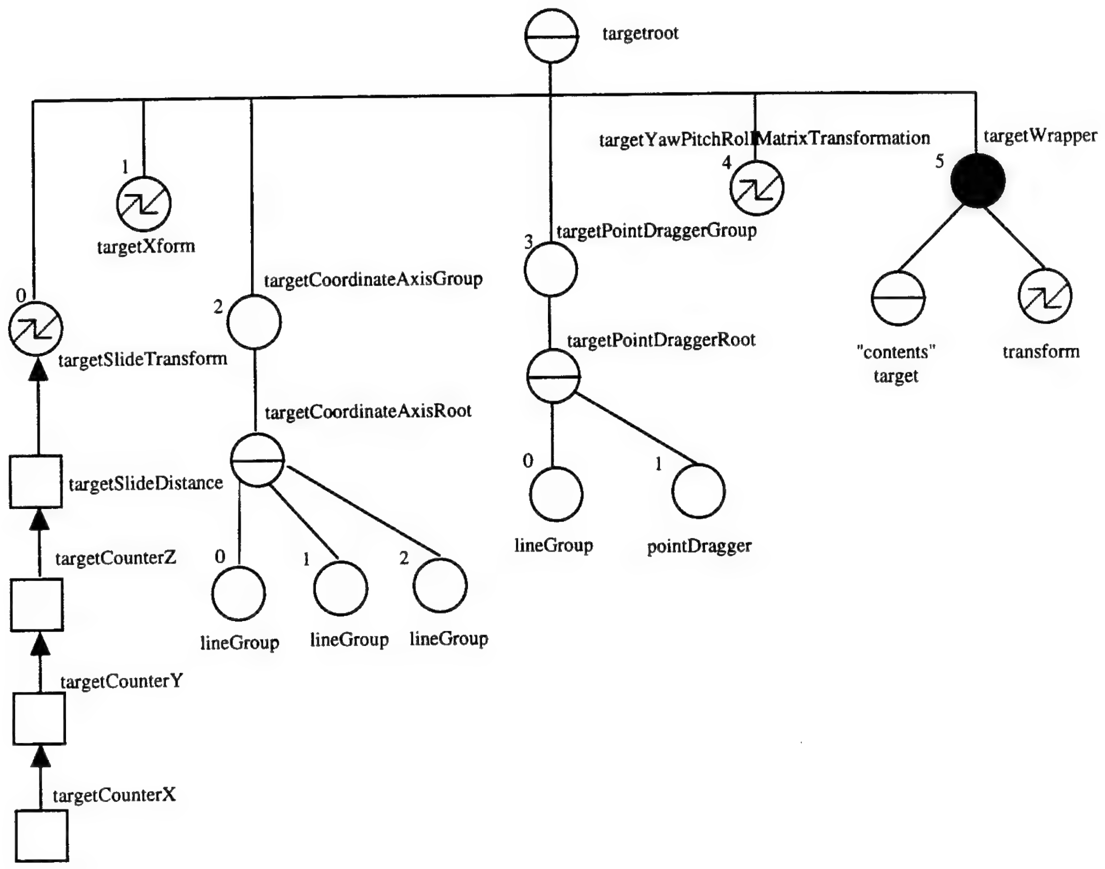


Figure A-3: AMES Target Database

A.4 . Fuze and Line Group Scene Database Subgraph.

Figure A-4 depicts the fuze cone's Inventor subtree and the lineGroup Inventor subtree. Reference Chapter 4 for a complete description of these trees.

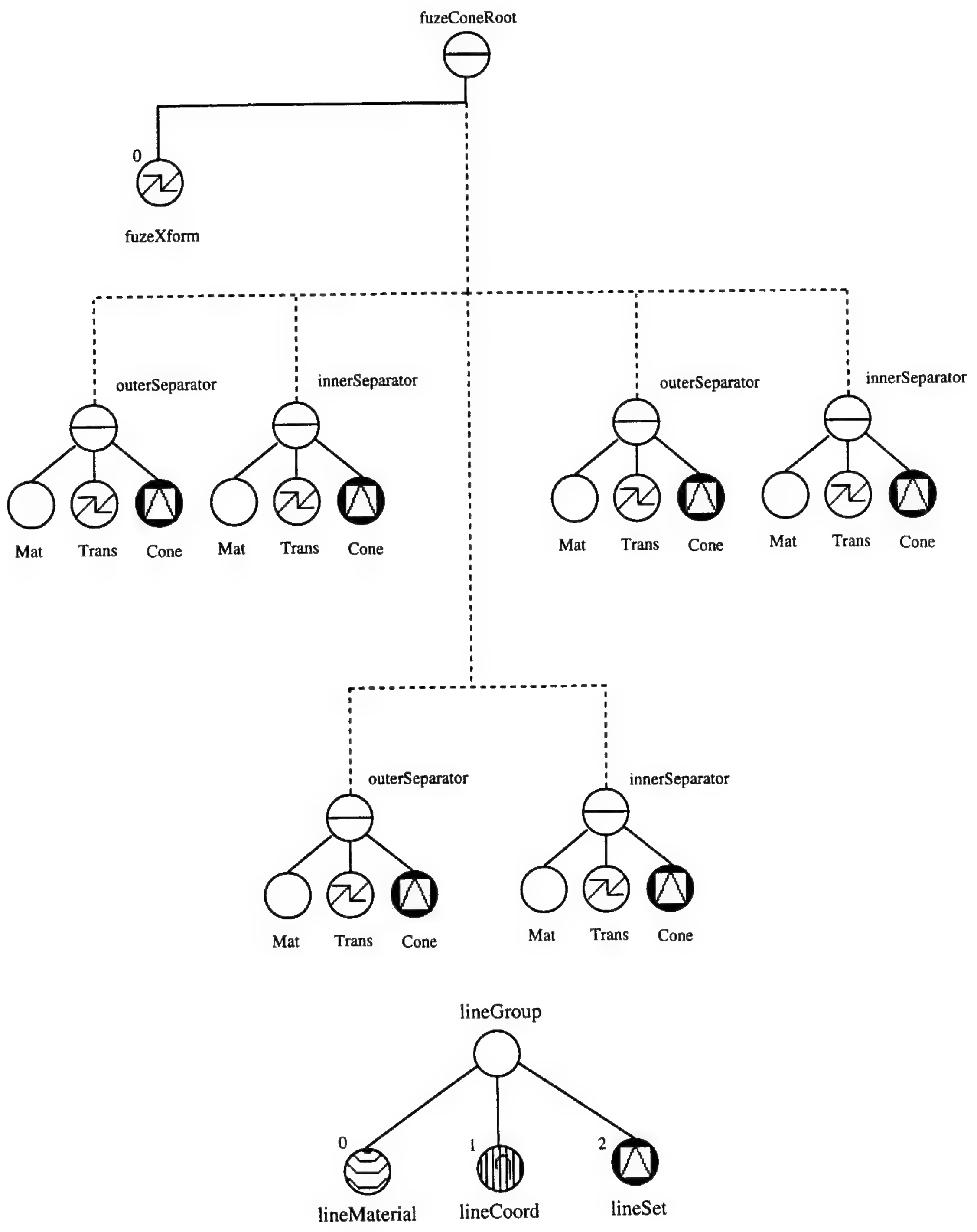
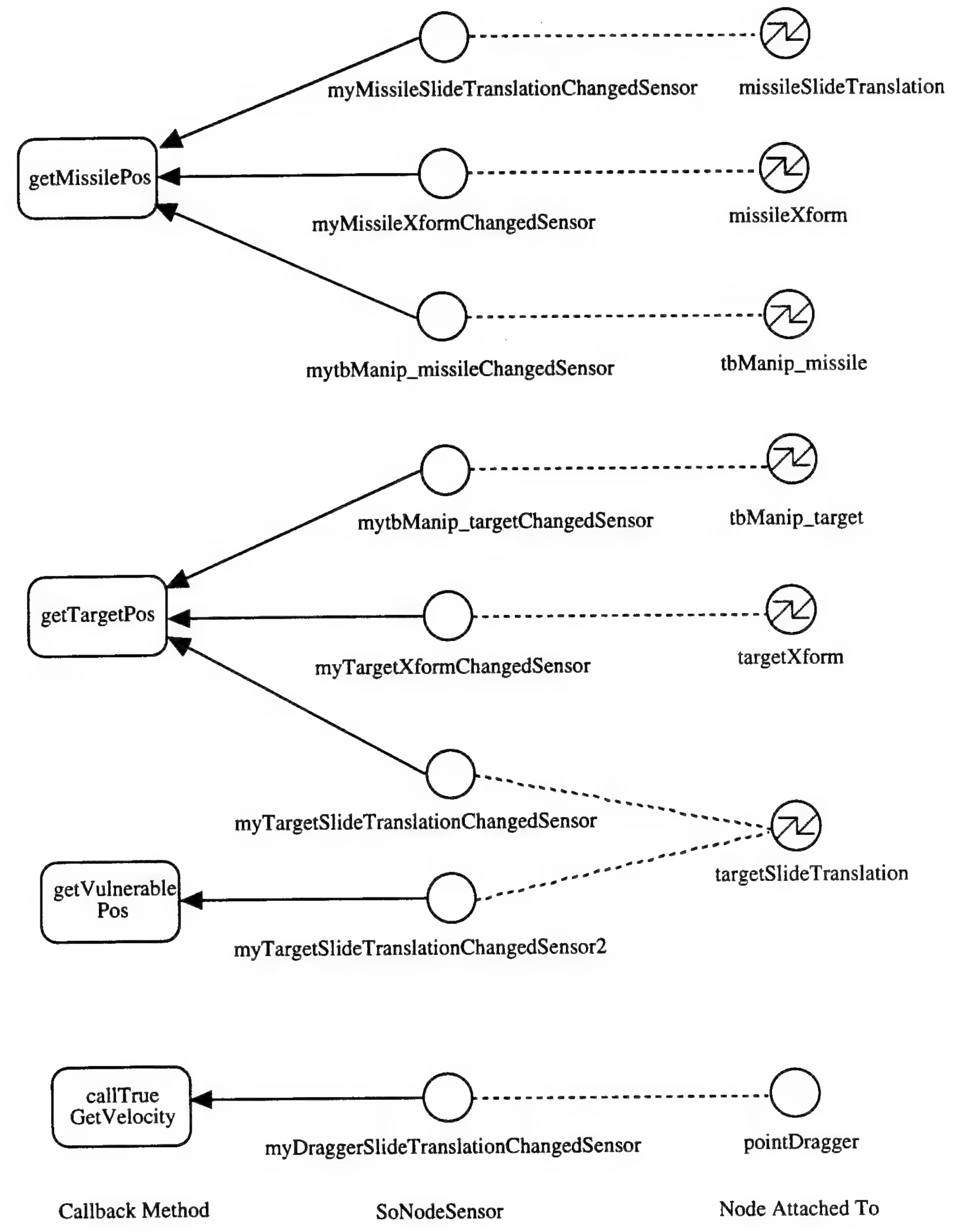


Figure A-4: AMES Fuze and LineGroup Database

A.5 . Callback Database Subgraph.

Figure A-5 depicts the connection of the SoNodeSensors used in AMES. It shows the Inventor node each sensor is attached to and the function called when the sensor is invoked.



myMissileSlideTranslationChangedSensor
missileSlideTranslation
getMissilePos
myMissileXformChangedSensor
missileXform
mytbManip_missileChangedSensor
tbManip_missile
mytbManip_targetChangedSensor
tbManip_target
getTargetPos
myTargetXformChangedSensor
targetXform
myTargetSlideTranslationChangedSensor
targetSlideTranslation
getVulnerable Pos
myTargetSlideTranslationChangedSensor2
callTrue GetVelocity
myDraggerSlideTranslationChangedSensor
pointDragger
Callback Method
SoNodeSensor
Node Attached To

Figure A-5: AMES Callback Database

Appendix B

B.1 . ENCOUNT Overview.

ENCOUNT is a FORTRAN program used to manipulate a missile and target FASTGEN file according to a set of endgame parameters included in an ENCOUNT style data file. The program directly manipulates each model file and creates a composite FASTGEN file of the given encounter.

B.2 . Directions for Running ENCOUNT.

ENCOUNT can be run on the SGI systems in the graphics lab but the only way to view the resulting FASTGEN file is to use Ivaview. Ivaview however, is a licensed product and can only be used on the SGI machines at WL/MNMF.

To run ENCOUNT, you must first create an ENCOUNT style data file. This data file is shown in Section B.3. The data file must be located in the same directory in which ENCOUNT is run. To run ENCOUNT, type *enc* at the prompt. This program is a shell script that asks various questions as to the FASTGEN files to be used for model input and the source data file as well. After entering the data, the *enc* script will automatically run *encount*. When finished, it will load the resulting FASTGEN file into Ivaview.

B.3 . Example ENCOUNT Input Data:

```
INPUT CONDITIONS FOR ALAMO/F106

MISSILE SPEED, TARGET SPEED, ATTACK AZ AND EL
1099.0  936.2   10.0  20.0

X, Y, AND Z OF AIMPOINT ON THE TARGET
-374.1    0.0    0.0

X, Y, AND Z OF THE TRACKING POINT ON THE MISSILE
```

```
 86.8   0.0   0.0

XMISS, YMISS, ZMISS
-125.0 -121.0 -101.7

YAW, PITCH AND ROLL OF THE TARGET
  0.0    .0   0.0

YAW, PITCH AND ROLL OF THE MISSILE
  0.0   0.0   0.0

X, Y AND Z OF THE CENTER OF ROTATION OF THE TARGET
-374.1   0.0   0.0

X, Y AND Z OF THE CENTER OF ROTATION OF THE MISSILE
  0.0   0.0   0.0

X, Y AND Z OF ARBITRARY POINT ON THE MISSILE
 -1.3   0.0   0.0

FLAGS FOR AXES
  1.0   1.0   1.0

FLAGS FOR TRAJECTORIES AND AIMPOINT SPHERE
  1.0   1.0   1.0  1.0
```

Figure B-1: ENCOUNT Data File

•

Appendix C

C.1 . Relative Velocity Calculation

To obtain the relative velocity vector, you must do a vector subtraction between the target velocity and the missile velocity as shown in Figure C-1.

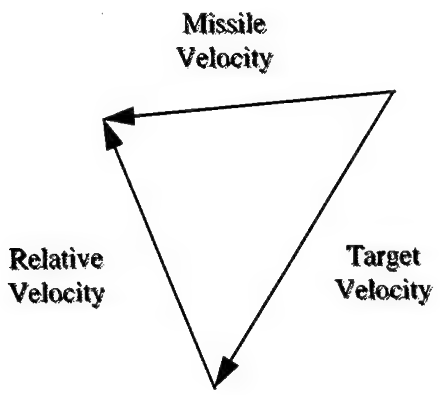


Figure C-1: Relative Velocity Vector Construction

C.2 . Relative Trajectory System

The relative trajectory system is created by taking a missile endgame scenario and assuming the target's velocity to be zero. With this assumption, the missile's velocity and direction are changed to create an equivalent scenario as the original that has both the target and missile moving. Figure C-2 shows the relative trajectory system's elevation and azimuth. It is these values that need to be calculated give the missile and target velocities to obtain the relative trajectory system. AMES calculates the relative trajectory system's elevation and azimuth according to the ENCOUNT code. For more information on how these calculations are made, please reference ENCOUNT.

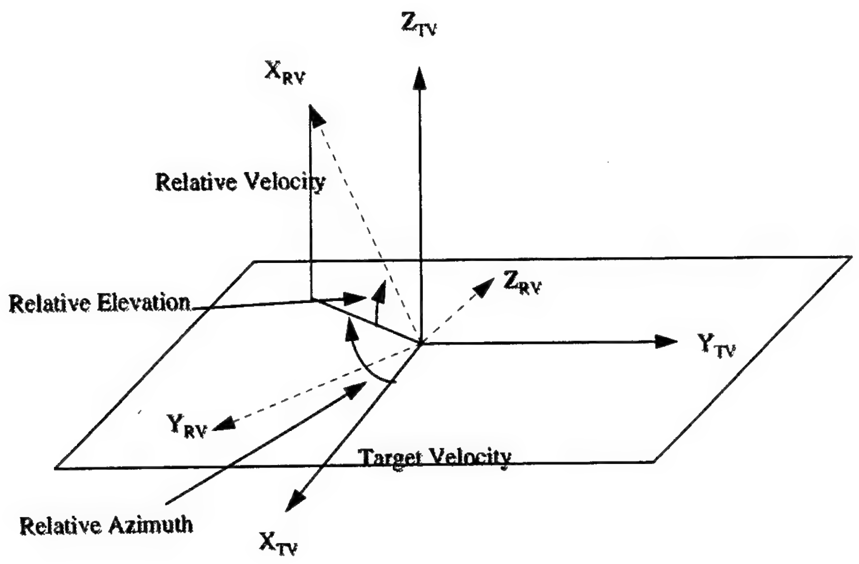


Figure C-2: Relative Velocity Coordinate System

Appendix D

D.1. AMES Snapshots.

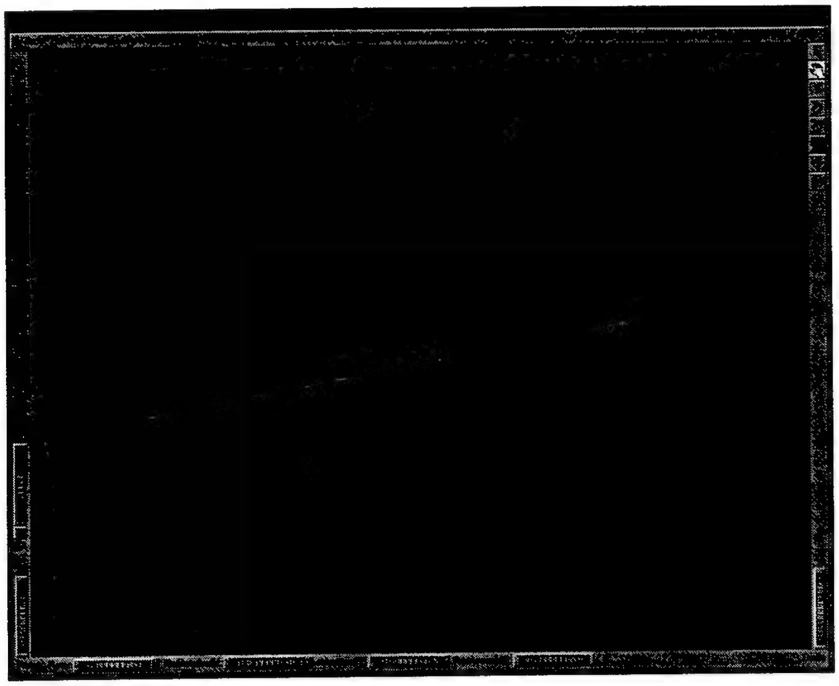


Figure D-1: Snapshot showing the relative trajectories and coordinate axes for an endgame scenario

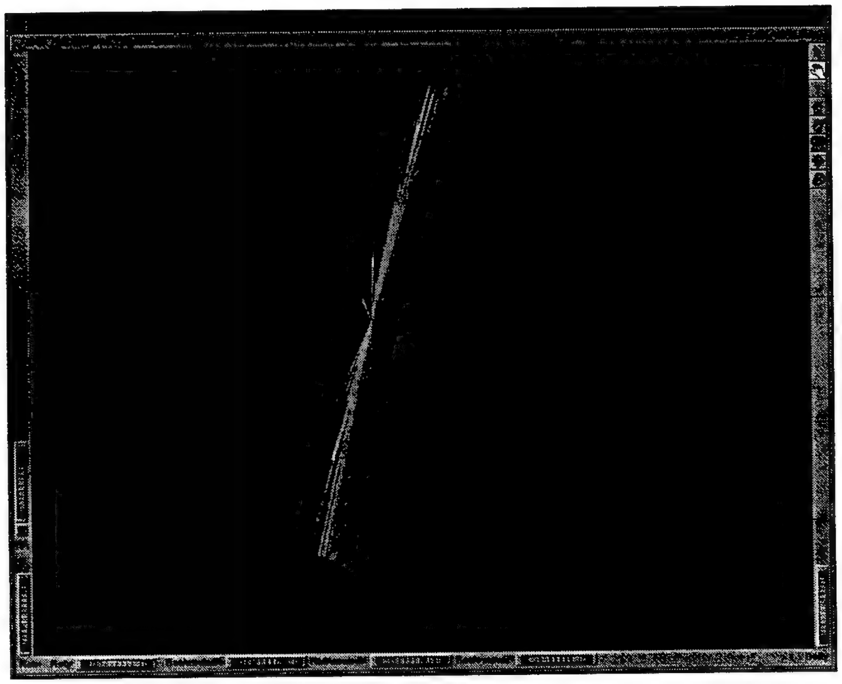


Figure D-2: Snapshot showing warhead fragmentation.

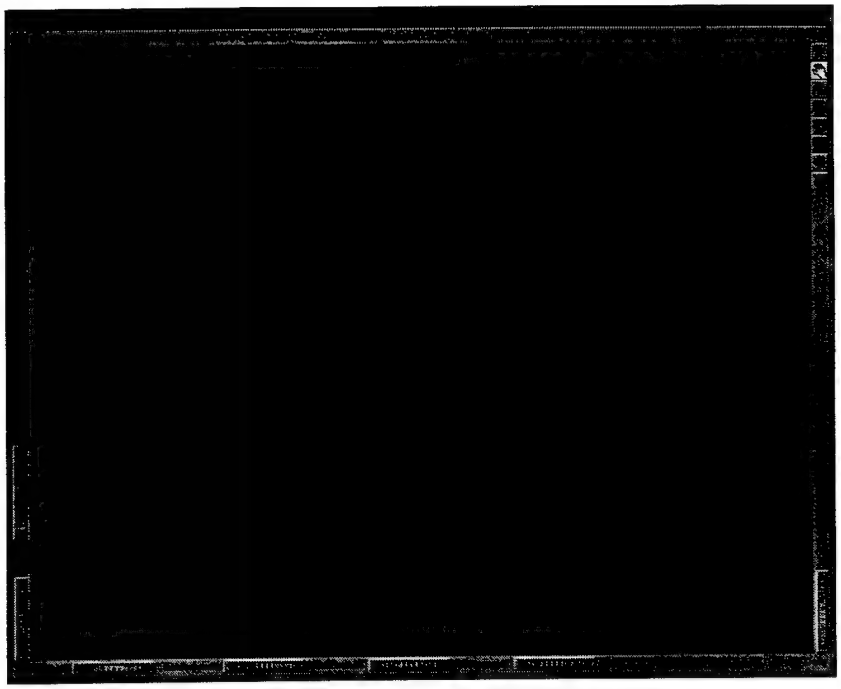


Figure D-3: Snapshot of the fixed fuze cone sensor coverage capability.

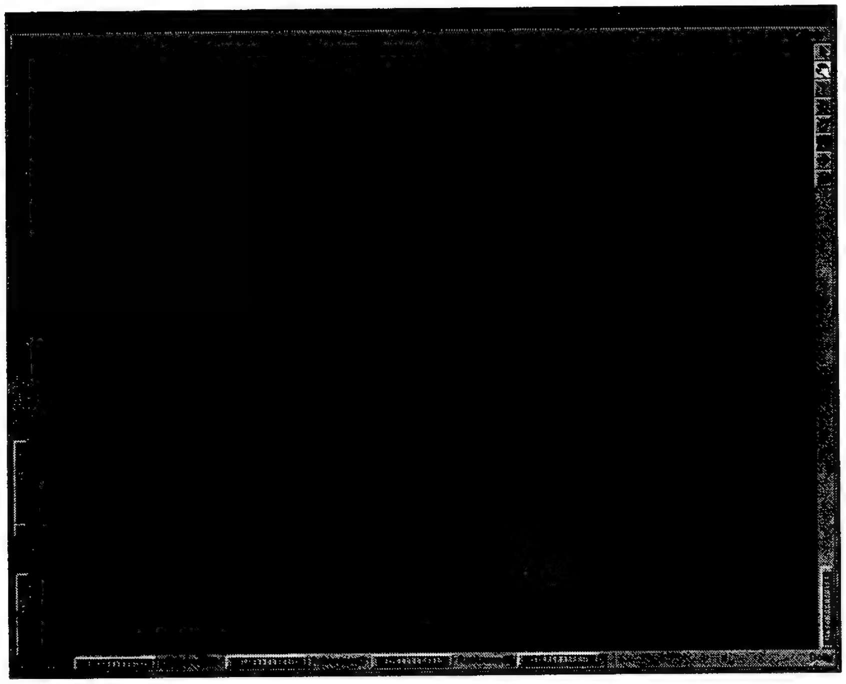


Figure D-4: Snapshot of the target damage capability.

Bibliography

1. Coffield, Patrick C. and others. User Manual for the Air-to-Air Effectiveness Program SHAZAM(U). Eglin AFB, Fl. : Defense Technical Information Center, ADB-104959L, 1986.
2. William K. Shirley. A Guide to FASTGEN Target Geometric Modeling. Fort Walton Beach: Defense Technical Information Center, ADA-273-171, 1993.
3. Douglas McGown. A Computer Program for the Graphical Illustration of a Missile and Target Encounter (GIMATE) User Manual. Defense Technical Information Center, ADB-110-553L, 1987.
4. PMC Inc. Ordnance Package Evaluation Code User's Manual, Socorro, NM, Unpublished.
5. Schroeder, William J. and others. Decimation of Triangle Meshes, SIGGRAPH '92 Conference Proceedings. 65-68. New York: ACM Press, 1992.
6. Josie Wernecke. The Inventor Mentor. New York: Addison-Wesley Publishing Company, 1994.
7. Josie Wernecke. The Inventor Toolmaker. New York: Addison-Wesley Publishing Company, 1994.
8. Open Inventor Architecture Group. Open Inventor C++ Reference Manual. New York: Addison-Wesley Publishing Company, 1994.
9. Cramer, Russel E. Hilbrand, Roy. FASTGEN 3 Target Description Computer Program. Washington, D.C.: Defense Technical Information Center, AD-B103-850, 1985.
10. The SURVICE Engineering Company. FASTGEN Ivaview User's Manual, Aberdeen, Maryland, Unpublished, 1992.
11. Captain Jeff Bush. AMES Decimation/File Conversion Project, Unpublished.
12. Doug McGown. ENCOUNT Program, Unpublished.
13. Silicon Graphics Inc. RapidApp User's Manual, Online Document.
14. Coryphaeus Software Inc. Designer's Workbench 3.1 User's Manual, 1995.
15. PMC Inc. ACE2VRML Program, Socorro, NM, Unpublished.
16. Silicon Graphics Inc. IRIS Viewkit Online Manual, IRIX 6.2, 1996.
17. Rumbaugh, James and others. Object Oriented Modeling and Design, New Jersey, Prentice-Hall Inc., 1991.
18. Jakob Nielson. Usability Engineering, London: Academic Press Inc, 1993.
19. Schroeder, William et. al. The Visualization Toolkit, New Jersey: Prentice Hall PTR, 1996.

20. Aerojet Electronics Systems Plant. MSFUZE program, Unpublished.

21. Berger, J. and Chouinard, L. SARA - An Object-Oriented Target Evaluation and Weapon Assignment Demonstrator, Proceedings of the 1994 Simulation Multiconference, April 1994, 29-34.

22. DeHaemer, Michael J., Zyda, Michael J. Simplification of Objects Rendered by Polygonal Approximations, Computers and Graphics, vol 15 num. 2, 1991, pg 175-184.

23. Franklin, Wm. Randolph, Kankanhalli, Mohan S. Parallel Object-Space Hidden Surface Removal, Computer Graphics, vol 24, number 4, Aug. 1990, pg 87-93.

24. Raghava G. Gowda. A Framework for Developing and Managing Reusable Avionics Software, Defense Technical Information Center, DTIC# AD-A276-846, September, 1993.

25. Hoppe, Hugues et.al., Mesh Optimization, Proceedings of the ACM, SIGGRAPH '93 Conference on Computer Graphics, 1993, pg 19-25.

26. Moore, Matthew, Wilhelms, Jane, Collision Detection and Response for Computer Animation, Proceedings of the SIGGRAPH '88 Conference: Computers Graphics, 1988, pg 289-297.

27. Nielsen, J. and Levy, J., Measuring Usability: Preference vs. Performance, Communications of the ACM, April 1994, Volume 37, Issue 4, 67-75.

28. Greg Turk, Re-Tiling Polygonal Surfaces, Computer Graphics Proceedings, SIGGRAPH '92 , vol 26, number 2, July 1992, pg 55-64.

Vita

Lt Joseph E. Moritz was commissioned in the USAF on 15 May 1993 after graduating from Swarthmore College with a degree in both Computer Science and Economics. His first assignment was temporary duty at Basic Computer/Communications Officer Training at Keesler Air Force Base, Mississippi. Upon graduation, he moved to Washington D.C. were he was assigned to the Air Force Pentagon Communcations Agency at the Pentagon. After 17 months on station, he received a scholarship to attend the Air Force Institute of Technology and entered the School of Engineering in May 1993.

Permanent Address: 16 School House Lane
Bridgeton, N.J. 08302

REPORT DOCUMENTATION PAGE	*Form Approved* *OMB No. 0704-0188*

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE December 1996	3. REPORT TYPE AND DATES COVERED Master's Thesis
4. TITLE AND SUBTITLE GRAPHICAL DISPLAY OF A MISSILE ENDGAME SCENARIO		5. FUNDING NUMBERS
6. AUTHOR(S) Joseph E. Moritz, Lieutenant, USAF		
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Air Force Institue of Technology 2750 P Street WPAFB, OH 45433-7126		8. PERFORMING ORGANIZATION REPORT NUMBER AFIT/GCS/ENG/96D-20
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) WL/MNMF Don Cunard 101 West Eglin Blvd, Suite 219 Eglin AFB, FL 32542-6810		10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES		
12a. DISTRIBUTION AVAILABILITY STATEMENT Approved for public release; distribution unlimited		12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 words)*

Traditionally, computer programs to model and simulate air-to-air missile endgame engagements have focused on improving the underlying probability of kill models. This focus, however, understates the benefits of current computer graphics technology for visualization and direct manipulation of missile endgame parameters and building engagement understanding through real-time, three dimensional simulation of the endgame. Our research has been to develop a versatile simulation system to display the missile endgame parameters and allow graphical interaction with these parameters. This program also allows the user to view animated engagements of previously designed endgames. The current project scope is to provide user feedback of multiple fixed cone fuzes, warhead fragment trajectories, and target damage incurred by warhead detonation given a particular set of missile and target endgame encounter parameters. Our implementation, the AFIT Missile Endgame Simulator (AMES), achieves its goals using a Motif/Inventor graphical user interface to change engagement parameters through both text input and direct object manipulation. It also allows these engagements to be animated over time to help develop intuition about the objects' interactions.

14. SUBJECT TERMS Computer graphics, modeling and simulation, missile endgame simulation			15. NUMBER OF PAGES 112
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT Unclassified	18. SECURITY CLASSIFICATION OF THIS PAGE Unclassified	19. SECURITY CLASSIFICATION OF ABSTRACT Unclassified	20. LIMITATION OF ABSTRACT UL